结构化程序设计教程

——True BASIC

姜遇姬 唐 全 顾玉珍等 编著

宇航出版社

923571
73.87424
C116
[illegible]
10357691

结构化程序设计教程

——True BASIC

姜遇姬　唐　全　顾玉珍等　编著

宇航出版社

（京）新登字181号

内 容 提 要

True BASIC语言是由BASIC语言的创始人推出的一种比较完善的、真正的BASIC语言系统，是不依赖机器硬件的典型的结构化程序设计语言，具有简单、小型、易学的特点和较强的计算、应用开发能力。

本书内容系统、全面、深入浅出，通俗易懂，实用性强。是一本学习和掌握True BASIC语言的较好的教材。

本书可作为高等院校计算机专业和非计算机专业的教材，也可供各部门计算机工作者、科研和工程技术人员学习和参考。

结构化程序设计教程
—True BASIC

遇姬·唐·全　顾玉珍等　编著

责任编辑：张国瑞

☆

宇航出版社出版
（北京和平里滨河路1号）
邮政编码100013

☆

南京虹达印刷厂印刷
新华书店北京发行所发行

开本：787×1092　1/16　印张：15.5　字数：390千字
1994年4月第1版　　印数：1—3000
ISBN 7-80034-656-0/TP·041
定价：14.80元

前　言

BASIC(Beginner's All-purpose Sybolic Instruction Code)表示初学者的通用符号指令代码，是美国 Dartmonth 学院的 John G. Kemeny 和 Thommas E. Kurtz 教授在1964年至1966年期间开发的小型程序设计语言。它具有简单、易学、小型的特点和人机交互计算功能，深受初学者的欢迎，广泛地用作微型计算机和小型计算机机内的基础语言，有些高级计算器也配置了 BASIC 语言。BASIC 语言由于编程、调试简单，又能编出功能很强的程序，化费较少的时间便可获得一个工作程序的快速响应能力，给计算机应用开发以及非计算机专业程序设计教学提供了良好的条件。BASIC 语言已广泛地应用于科学计算、数据处理、自动控制、辅助设计与制造、绘图、仿真、游戏等多种领域，是应用最广泛、最受欢迎的程序设计语言之一。

微电子技术和计算机硬件、软件技术的发展，尤其是微处理器和微型计算机的问世，一方面给 BASIC 语言的应用开辟了更广泛的前景，但另一方面，BASIC 语言的一些缺陷亦随之暴露。例如，多数 BASIC 语言采用解释执行方式，对大量数据的数值计算或多次运行的程序效率低、适应性差；由于运行态查错，以及出错的致命性，使用户实用时感到不十分方便；BASIC 是非结构化的程序设计语言，很难用它编写出完整的结构化、模块化的程序，对现代程序设计技术的适应性差；BASIC 语言的功能已不能满足科学技术发展的需要。因此，计算机厂商为争夺市场，发扬各自机器的优势，纷纷改进和扩展 BASIC 语言的功能，从而出现了众多的与机器硬件相关的 BASIC 语言版本，人们称之为 BASIC 的“方言”。BASIC 的“方言”严重地破坏了它的通用性和程序移植性，给 BASIC 语言的学习、应用带来了困难。

True BASIC 是一种真正的 BASIC 语言，由原 BASIC 的创始人 J. G. Kemeny 和 T. E Kurtz 于1986年推出。它是严格符合美国国家标准、不依赖机器硬件的典型结构化程序设计语言。它保留了原 BASIC 语言易懂、易学的基本特点，排除了原来的一些缺陷，吸取了 FORTRAN、PASCAL 语言的优点，从而具备了过去大、中型计算机才具备的各种较强的计算功能，具有足够的能力适应较大课题的开发。

True BASIC 具有下列特点：

(1) 不依赖于硬件，可以在任何微机上运行，编写的程序具有移植性；与原 BASIC 语言有互通性，对原 BASIC 用户无需了解语言系统中存在的差异，

也不必再学习 True BASIC 语言。

(2) 无 GOTO 语句的结构化程序设计语言系统，控制结构丰富、灵活，完全能够适应现代结构化、模块化程序设计技术的要求。

(3) 提供了解释执行和编译执行两种方式，兼顾了初学者交互学习、使用和应用、开发两者的需要。解释方式的跟踪能力强，便于程序纠错、调试；编译方式程序运行效率高。在 2.03 版本中，还提供了产生 DOS 系统可直接执行 EXE 文件的软件包。

(4) 提供了图形、音乐功能，图形语句与像素无关，并支持丰富的彩色、图形输入、子图几何变换和动画制作；提供了外部函数、子程序和子图结构，以及便于用户建立、使用的高效程序库；提供了两种格式文件和自由格式的字节流文件，满足对任意结构数据文件处理的要求；可充分利用大于 6K 内存，……。这一系列扩展、增强的功能，无论对初学者或是训练有素的程序员都有极大的吸引力。

(5) 配备了功能很强的屏幕编辑系统，用户可在全屏幕编辑的环境下，方便地编辑、修改原程序；提供了系统自助功能，在需要时，用户可获得系统的帮助。

为了满足大专院校非计算机专业基础教育和 True BASIC 程序设计的需要，我们编写了这本教程。采用通俗性、科学性和实用性结合的方法，以语言是算法实现工具的原则，通过大量例题，深入浅出地、系统地介绍了 True BASIC 语言和 True BASIC 语言程序设计方法和技术。此外，本教程还包括计算机基础知识、微机基本操作、上机使用实验指导和必要的附录。

本书由姜遇姬、唐全任主编，顾玉珍任副主编，参加编写的有朱敦名、林可裕，全书由胡汉才主审。第 1、2、9 章由唐全编写，第 3 章由林可裕编写，第 4、6 章由顾玉珍编写，第 5 章由朱敦名编写，第 7、8、10 章由姜遇姬编写。

由于编写者水平有限，错误和不足之处，恳请读者批评指正。

编　者　　1993. 8.

目　　录

第 1 章　电子计算机基本知识

1.1　计算机的发展 …… 1
1.2　计算机的用途 …… 2
1.3　计算机硬件基本组成 …… 4
1.4　计算机语言与计算机软件 …… 5
1.5　计算机中采用的计数制 …… 7
1.6　用计算机解决实际问题的过程 …… 12
习题 …… 13

第 2 章　微型计算机基本操作

2.1　微型计算机的发展 …… 14
2.2　操作系统和 DOS 的组成与结构 …… 16
2.3　磁盘与磁盘驱动器 …… 18
2.4　文件系统概念 …… 19
2.5　常用的 DOS 命令 …… 22
习题 …… 29

第 3 章　数据与数据操作

3.1　常量与变量 …… 30
3.2　数值运算与数值表达式 …… 32
3.3　字符串表达式与字符串操作 …… 36
3.4　变量的赋值与数据输出语句 …… 37
3.5　True BASIC 源程序 …… 40
3.6　键盘输入数据语句 …… 43
3.7　DATA/READ 语句与数据表恢复语句 …… 46
3.8　输出位置与数据格式控制 …… 49
习题 …… 57

第 4 章　判断选择结构与循环结构

4.1　逻辑条件与逻辑表达式 …… 59
4.2　判断选择结构 …… 62
4.3　多分支判断选择结构 …… 67
4.4　计数循环结构 …… 71
4.5　条件循环结构 …… 77
4.6　循环嵌套结构 …… 86
习题 …… 89

第 5 章　数组与数组运算

5.1　数组的概念 …… 92
5.2　数组输入输出 …… 97

5.3 排序与检索 …… 107
5.4 数组应用 …… 111
习题 …… 113

第6章 函数与子程序

6.1 标准函数 …… 115
6.2 自定义函数 …… 120
6.3 子程序 …… 128
6.4 程序库文件 …… 137
6.5 出错处理与出错函数 …… 144
习题 …… 149

第7章 文件

7.1 MS-DOS 文件 …… 151
7.2 有关文件操作的系统命令 …… 152
7.3 数据文件 …… 153
7.4 正文文件 …… 157
7.5 记录文件 …… 163
7.6 字节文件 …… 168
习题 …… 170

第8章 图形与声音

8.1 图形环境与作图语句 …… 171
8.2 函数曲线图与多窗屏幕设置 …… 178
8.3 子图与子图变换 …… 185
8.4 BOX 语句与动画 …… 187
8.5 声音 …… 191
习题 …… 195

第9章 结构化程序设计方法

9.1 程序与程序设计 …… 196
9.2 开发程序系统的全过程 …… 200
9.3 程序系统中错误类型分析 …… 201
9.4 程序调试与测试方法 …… 204
9.5 结构化程序设计方法 …… 206
习题 …… 213

第10章 使用与实验指导

10.1 MS-DOS 操作系统使用实验 …… 214
10.2 True BASIC 程序设计 …… 217
附录1 ASCII 码符号集 …… 234
附录2 出错信息代码及解释 …… 235

第1章　电子计算机基本知识

电子计算机的出现和发展是本世纪科学技术的卓越成就之一。由于计算机运算速度快、存储容量大、有“记忆”能力和逻辑判断能力，并能按程序要求高精度地完成计算或控制任务，所以已广泛地应用于科学计算与工程设计、实时控制与工业自动化、信息数据处理与人工智能等多个领域。计算机能够完成很多人们不能想象也无法完成的任务。可以说没有计算机就根本谈不上今天的现代化。同时，现代科学技术和生产技术的发展，反过来又推动了计算机技术的发展及计算机技术的应用和普及，现在人们还在不断地为计算机开拓新的应用领域。今天的计算机技术仍以旺盛的生命力突飞猛进地向前发展着。

为了更有效地学习和应用计算机技术，我们首先要对计算机的发展和用途、计算机的硬件和软件、计算机采用的计数制和计算机解决实际问题的过程有个初步的了解。

1.1　计算机的发展

1.1.1　人类工具的发展

生产的需要，科学的发展，使人类不断地制造出新的工具并利用这些工具和大自然作斗争，为人类的生存和生活服务。人类工具的发展大致可分为以下几个阶段：

(1) 原始社会里人类制造了刀和斧等工具。刀和斧等工具的出现是人类四肢的延伸，它使人类和大自然斗争有了原始工具，并增强了人类劳动的本领。

(2) 人类制造出蒸汽机、电动机。蒸汽机和电动机的产生，引起了产业革命，大大促进了工业的发展，它的出现是人类肌体的延伸。

(3) 无线电、电视机时代。人类制造了无线电、电视机，可以快速地传递信息，加速了通信现代化，克服了大千世界时间、空间的障阻。无线电、电视机的产生，使人类感知器官得到了延伸。

(4) 电子计算机时代。由于动力的利用，产生了机械计算机。电子工业的发展使电子计算机的制造成了现实。电子计算机的出现使人类进入了信息社会时代。电子计算机的应用是人类大脑的延伸，它大大减轻了人类的脑力劳动。

(5) 智能型计算机。人工智能系统的实现是人类思维的延伸，它标志着智能型计算机时代的到来。现在，多媒体技术、神经元技术的出现加速了新一代计算机的发展。新一代计算机大大促进人类生活物质文明和精神文明的进步。

1.1.2　计算机的诞生和发展

人类在生产实践和科学研究中，发展了电子技术，这时更迫切需要有计算速度快、精确度高、并能按程序规定自动进行计算和控制的新型工具，这样电子计算机就应运而生了。实际上，电子计算机的出现是现代科学技术发展的必然产物。1946年，美国宾西法尼亚大学的J. W. Mauchy 和 J. P. Eckert 研制成功了世界上第一台计算机 ENIAC(Electronic Numerical Integrate And Calculate)。ENIAC计算机由18000个电子管、1800个继电器组成，耗电150千

瓦，占地 1800 平方英尺，重达 30 吨，但运算速度每秒仅 5000 次。ENIAC 计算机在美国陆军弹道研究所运行了 10 年，主要用于弹道计算。尽管这台计算机体积大、速度慢、耗电多、性能差，但它的诞生开辟了人类文明的新纪元，使人类开始进入了电子计算机时代。

从 ENIAC 第一代计算机的诞生到今天已近半个世纪了，这在人类历史长河中仅仅是短暂的一瞬间，但计算机硬件(Hardware)的发展已经经历了电子管、晶体管、集成电路和大规模集成电路四代，现在正向超大规模集成电路智能型机发展。由于计算机系统软件(System Software)和应用软件(Application Software)的不断开发和应用，计算机已由过去无监督程序(Moniter)或管理程序(Supervisor)、更无操作系统(Operating System)的“裸机”，发展到今天具有丰富软件的现代化计算机。今天的计算机不仅具有功能齐全的操作系统、各种语言的编译程序、使用方便的编辑程序，而且有各式各样的应用程序供用户使用。

1.1.3 电子计算机的展望

目前，电子计算机正朝着微型化、巨型化 、智能化和网络化方向发展。

(1) 微型机。由于微型机性能可靠、使用方便、价格低、功能强，因而已得到推广和普及。当前微型机一方面面向小型、轻便、可携带式笔记型计算机发展，另一方面已逐步升级至超级微机、工作站。华胜 2000 型就是典型的性能良好的国产计算机工作站。

(2) 巨型机。计算速度可达数十亿次至数百亿次，主要用于天气预报、航空航天、军事科学等部门。我国国防科技大学 1992 年生产的银河 I 号巨型机，速度可达每秒 10 亿次，现在正在气象预报部门运行。1993 年，银河仿真— I 型计算机通过了国家鉴定，该机研制起点高、跨度大、技术先进，整体技术性能达到国际水平，标志着我国已经跨入了世界仿真机先进国家的行列。

(3) 计算机网络。若干台本来独立的计算机通过通信线路和网络软件互相联接起来，组成计算机网络，使网上的用户可以共享资源。网络以实现远程信息处理为主要目标，它具有数据传输、负荷均衡等特殊功能。

(4) 智能型计算机。智能领域专家系统、管理决策支持系统、图像理解、语言识别等学科的发展，急需智能型计算机。智能型计算机能模拟人的智能，具有学习、思维、推理和决策功能。现在的机器人就是一种初级智能型计算机。智能型计算机的研制和设计从根本上改变了冯·诺伊曼计算机原理，采用知识存储、决策信息输出原理，它将带来计算机硬件和软件的革命，意义极其深远。智能型计算机能够解决那些至今人们还不知道如何解决的问题，从而大大扩充了计算机的用途。

可以说计算机的出现是一个划时代的事件，所以有人把蒸汽机、电动机、计算机的出现称之为三次工业革命。

计算机发展的总趋势是体积越来越小，性能价格比越来越高，存储容量越来越大，功能范围越来越广，功耗越来越少。现在既出现了运行速度高达数百亿次的巨型机，也出现了使用方便、运行可靠的大量的微型机。可以预料，不久的将来，电子报纸、电子图书等会呈现在我们的面前。

1.2 计算机的用途

时代的进步和科学的发展需要信息，信息依靠计算机，计算机在于应用。众所周知，计算

机已成为当今世界上最重要、最先进的一种计算和控制工具，它已深入到人类生活中几乎一切领域，如工业、农业、国防、教育、医药卫生、商业、交通以至家庭生活等各个方面。21世纪将是全面信息时代，人类从事的各项活动都离不开计算机系统的应用。这里从以下几个方面来讨论计算机的应用情况。

1.2.1 科学计算

电子计算机最突出的优点是高速度、高精度，因此它最适合于处理科学技术中的复杂模型，进行科学计算和工程设计。有些计算问题不但复杂而且有很强的时间性，不用计算机就不可能及时完成任务。例如天气预报的数据处理，用人工计算或手摇计算机进行计算，一般需要好几天才能完成，即使是计算、分析准确也失去了“预报”的实际意义。洲际导弹的发射与跟踪，反导弹技术的应用，都需要随时发现、跟踪、并指挥拦截导弹。总之，这些新技术都离不开计算机的应用。

上海文化广场的钢架屋顶钢管拉力的计算，需要解算包括107个未知数的联立方程，用手摇计算机要100个人算8个月，而用电子计算机只要几分钟就能精确地算出结果。现在设计一个球型钢架建筑要解几千个联立方程组，一个人可能计算一辈子还不一定能算得准确，而用现代的计算机仅需一二个小时。据说美国现有计算机完成的工作量，需要4000亿至5000亿人才能够完成。

1.2.2 数据处理和信息加工

利用计算机存储容量大、具有记忆能力和逻辑判断能力的特点，可以对大量的数据进行处理和信息加工。在企事业的金融管理中，用计算机进行分类汇总、成本核算、仓库管理、市场分析、计划统计、分析决策等；在铁路、航空等交通部门用计算机进行自动票务管理、交通监测和交通管制与调度；在医药卫生系统，计算机用于医院药房、病历、血库的管理；CT断层诊断系统，就是利用计算机对X射线的反射波进行数据处理后，给出人体内部不同图像，从而用于病情诊断。

办公室自动化实际上是信息处理过程自动化，它包括信息的输入输出自动化和信息存储加工过程自动化。办公室的事务实际上就是信息管理和信息加工。人事档案管理、情报资料管理、人口信息管理等，只有利用计算机才能提高效率。一个智能化的管理系统实际上就是一个决策支持系统。

1.2.3 实时过程控制

计算机加上敏感的检测元件和控制机构，就可以用于实时过程控制。不少企业生产采用计算机控制，提高了产品质量，降低了成本，产品得以升级换代，工作效率和经济效益都大为提高。自动控制系统利用了计算机实时、快速采集功能和信号处理的能力，为工业生产和实现大型化、综合化、高速化和自动化管理创造了条件。

在现代化钢铁厂中，一台带钢冷连轧机或热连轧机（1700mm轧机）改用计算机控制后，产量是人工控制的100倍，而且钢板的宽度、厚度等质量要求都有明显提高。现在机械加工领域的数控机床、化工生产的过程控制、航天飞行器的监测和遥控、医学上的各种自动化测试仪器等，都是应用计算机进行实时控制。

1.2.4 人工智能

人工智能（AI）的目的就是希望计算机能模拟人类的智力活动，具有学习、推理和思维的功能。人工智能是计算机学科中的热门课题，研究、应用的领域有：机器人——利用有对策能

力的机器博奕、机器学习和机器识别等；自然语言理解——研究机器“理解”和“创造”句子，传达语言信息；定理证明——研究机器逻辑推理；专家系统（ES）——医疗专家系统、找矿专家系统、故障诊断系统等。

现在人工智能方面比较成熟的是专家系统。例如，质谱法 Dendral 专家系统，它是美国 Stanford 大学用计算机解决有机化学中结构解析问题的科研项目。它用一台带有计算机的质谱仪，用电子流轰击化合物分子，使分子中连接各个原子的某些键发生断裂，并把有关信息记录在质谱仪上，再用计算机进行分析和数据处理，最后根据分子式便能得出未知化合物的分子结构。启发式 Dendral 专家系统利用计算机实现了化学专家水平的推理与决策，把这些信息存入专家知识库、专家规则库，然后根据用户提出的问题和 Dendral 系统进行对话来实现推理决策。

1.2.5 计算机辅助系统

由于计算机硬件设备和图形设备不断更新和扩充，系统软件和图形软件功能不断完善和提高，使计算机图形学在 80 年代得到了更加广泛的应用。其中使用最广泛、最活跃的领域是 CAD（Computer Aided Design）和 CAM（Computer Aided Manufacturing）技术。CAD/CAM 是计算机应用领域的重大开拓，它的应用十分成功，并已取得了巨大的成果。同济大学 CAD 中心，在 SUN 工作站上安装了 DOGS（Design Office Graphics System）图形软件系统，它对上海南浦大桥的建设在方案选择、图纸施工、测量计算和决策管理中都起了重大作用。

CAD 的特点是设计灵活、直观，设计周期短，效率高，易于实现局部修改和诸方案优化决策等优点。世界上工业发达国家竞相开发 CAD/CAM 系统，他们希望自己在产品技术上领先，占领国际市场并取得更大的经济效益。现在，国内外已推出一些适合各种层次的计算机 CAD/CAM 系统，并在机械、电子、建筑、航空、船舶、汽车、化工、纺织、服装等行业中普遍采用 CAD/CAM 技术。

目前，常用的计算机辅助系统还有计算机辅助教学（CAI）、计算机辅助测试（CAT）等。计算机仿真系统（Simulation system）可以模拟核反应过程、飞行器飞行和企业的生产管理。计算机辅助系统还可应用于办公室和家庭生活、激光排版、电子邮件和自动通信等方面，大大改变了人们生活的现状与面貌。

由此可见，计算机的应用已经硕果累累。学习计算机的目的在于应用，而应用的需要又是一门学科得以发展的最重要的推动力。计算机学科已成为内容无穷尽、发展无止境的一门技术先进、应用广泛的新兴学科。

1.3 计算机硬件基本组成

一台计算机典型的硬件系统通常由运算器、控制器、存诸器、输入和输出设备等四部分组成，如图 1.1 所示：

（1）运算器。运算器是计算机进行运算的部分，是快速的电子部件。它根据程序中指令规定的功能，在控制器的控制下完成加、减、乘、除等算术运算和逻辑运算。运算器相当于人工计算的算盘。

（2）控制器。计算机的各个部件在控制器的控制下，按照程序指令的要求协调动作。控制器是计算机的“神经中枢”，相当于人工计算的手和脑。执行什么操作，什么时候取数，从什

么地方取数，进行什么运算，算完后结果送到哪里等等，这些动作都是由控制器指挥运算器、存储器、输入和输出设备自动地、有条不紊地进行的。

(3) 存储器。存储器是指内存，用来存放程序、数据及其他信息。它是由一个个单元组成的，每个单元可以存放一个数据或一条指令。每个单元有一个地址编号，便于用指令来存取信息。计算机要进行计算，事先必须将计算步骤和有关数据存储起来，然后按规定的步骤进行计算，计算过程中还要保存中间结果和最后结果。存储器是计算机的记忆装置，相当于人工计算的纸和笔。

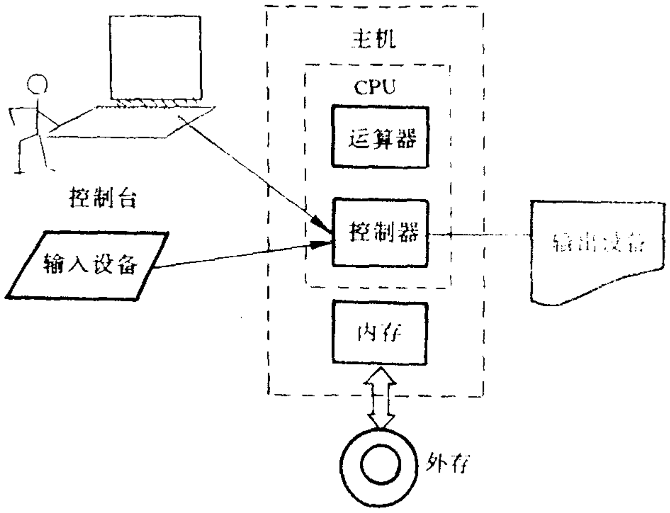


图 1.1　计算机硬件基本组成示意图

(4) 输入和输出设备。计算机只有上述三种设备是无法工作的，因为计算机要进行科学计算和过程控制，人们应事先把规定的计算步骤和原始数据送给计算机，而计算结果又要从计算机里取出来。这种人和计算机联系的桥梁称为输入和输出设备，常用 I/O 设备来表示。常用的 I/O 设备有键盘、光笔、鼠标、打印机、终端显示器 CRT(Cathode Ray Tube)、绘图设备、数字化仪以及各种磁介质设备等等。

计算机的运算器、控制器合称为中央处理机 CPU(Central Processor Unit)。计算机的主机包括中央处理机和存储器(内存)。输入设备和输出设备统称为计算机的外部设备。另外，存储器除了内存储器之外，还有外存储器，它包括软盘、硬盘和磁带机。外存储器可以长期保存大量的数据、程序和各种信息。内存储器中的信息可以输出到外存储器中，外存储器中的信息又可以直接送给内存储器，因此，磁带设备和磁盘设备也可作为输入和输出设备。

1.4　计算机语言与计算机软件

1.4.1　计算机语言

简单地说，人们的语言是人们交流思想的工具，谈话(声音)、书信(文字)就是人们的语言。电子计算机是人们用来进行科学计算、数据处理的工具，要让计算机为我们做事情，就得设法把我们的“意思”用特定的“信息”告诉计算机，也就是把如何计算或如何控制的信息传达给计算机。这种“信息”实际上就是人们和计算机联系的一种“语言”，这种语言称之为计算机语言。一般说来，计算机语言有机器语言、汇编语言和高级语言。

(1) 机器语言。一台机器指令(Instruction)的集合就是机器语言(Machine Language)。例

如，要进行 8 和 10 相加运算，IBM-PC 机的机器指令是：

```
10110000（操作码）    00001000（操作数）    将 8 送累加器 AL
00000100（操作码）    00001010（操作数）    AL 中的 8 和 10 相加，结果存 AL 中
```

显然，机器语言可在机器上直接运行，速度最快，但它不易理解和记忆，难检查、难调试修改，移植性差。

（2）汇编语言。用助记符代替机器指令，这就是符号语言，又称为汇编语言（Assembler language）。上面的两条机器指令可以写成下列汇编指令：

```
MOV    AL ,   8 ;
ADD    AL ,   10 ;
```

这是用汇编语言写的源程序（Source Program）。源程序要通过系统软件汇编程序转换为机器语言程序，即目标程序（Object Program），计算机才能执行。图 1.2 是汇编和执行过程示意图。汇编语言比机器语言易于理解且编写程序较方便，但它仍和机器语言一样是面向机器的“低级语言”。

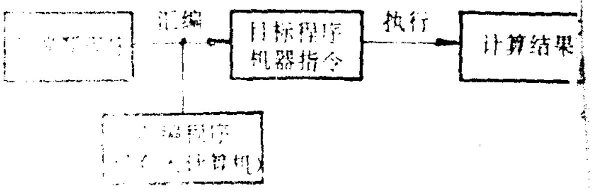


图 1.2 汇编源程序运行过程

（3）高级语言。为了推广计算机的应用，让众多的非计算机专业人员来编写程序、使用计算机，人们创造了高级语言（High Level Language）。这种语言就象人们说的话、写的文章，所以比较简单、形式化，不象人们的语言那样生动活泼、自然优美、丰富多彩。高级语言是通用的程序设计语言，不仅比汇编语言好学，而且可用于不同类型机器，解决不同问题，所以人们称高级语言是面向问题的语言。高级语言接近人们的术语和数学公式，便于人们学习和采用，可减轻程序设计工作量，为计算机的推广应用开辟了广阔的途径。

高级语言种类繁多，总的说有通用和专用两类。通用语言使用较多的有 BASI FORTRAN、PASCAL、COBOL、C 以及 PROLOG 和 LISP 等。BASIC 语言特别适用于初学者在计算机专家的不断改进下，功能越来越强，结构化程度越来越好。如 True BASIC 已成为机上最重要的语言之一。专用语言很多，如船体放样语言、地震勘探语言等，它们适用于特定的 CAD 行业中。

用高级语言编写的源程序机器是不能“识别”的，必须翻译成目标程序才能执行。将序翻译成目标程序，一种是用解释方式，另一种是用编译方式。解释方式是系统软件解释程序逐句地对源程序进行解释，解释一句，执行一句，即边解释边执行。编译方式是编译程序把源程序全部翻译成目标程序之后，再直接运行目标程序。解释方式和编译方式的执行过程如图 1.3 和图 1.4 所示。解释方式比编译方式花费的时间多，但易于对话，调试修改方便。

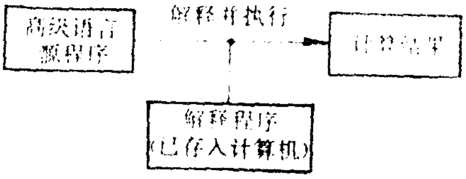


图 1.3 解释方式工作过程

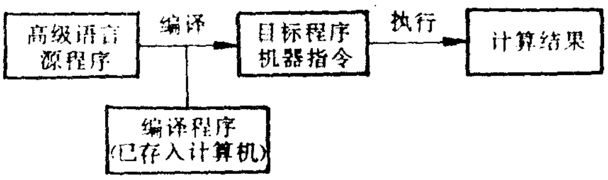


图 1.4 编译方式工作过程

早先的 BASIC 语言大多数是解释型的，而 True BASIC 语言是解释、编译两种方式通用，这为用户提供了方便。

1.4.2 计算机软件

我们已经知道，CPU、存储器和外部设备是计算机硬件。硬件是摸得着、看得见的实实在在的东西。那末，计算机软件又是什么呢？软件是计算机的各种程序和资料文档的总称，它们是看不见、摸不着的东西。可以这样说，硬件是计算机的“物质基础”，软件是计算机的“上层建筑”；硬件是计算机的“躯体”，而软件才是计算机的“灵魂”。

硬件和软件的概念已扩充到很多领域。有人说，人本身也是“硬件”和“软件”的结合，人的躯体是“硬件”，人在成长过程中不断学习和掌握的各种知识和技能，逐步树立起来的世界观和人生观，就是人的“软件”。

计算机软件分为系统软件和应用软件。系统软件是厂家提供的，如操作系统、各种语言的编译系统、机器诊断测试系统等。操作系统是管理、指挥计算机各个部分（硬件和软件）协调工作的程序系统。操作系统的种类较多，如单用户的早期DOS系统，多用户分时UNIX操作系统。XENIX系统是为适应80年代以来微型机的发展、普及而对UNIX进行修改和扩充后，安装在高档微机上的一种操作系统。应用软件除文本编辑、数据管理、图形处理程序外，还有用户根据自己需要而编写的一些专用程序，如药房管理程序、药物配伍禁忌查询程序等。由于共同的需要，常把同类型的专用程序整理集中在一起，组成相应的软件包，如把医药领域中常用的统计分析程序集中起来，成为医药学统计分析软件包。

计算机软件是脑力劳动的成果，它是有价值的，可以象商品一样出售。随着计算机日益广泛的使用，软件需求量越来越大，开发软件已逐渐成为一门新兴产业。

计算机是一个复杂的系统，它是由硬件和软件组成的，图1.5所示为计算机系统的组成。事实上，计算机硬件的功能一定要通过软件来实现和发挥。

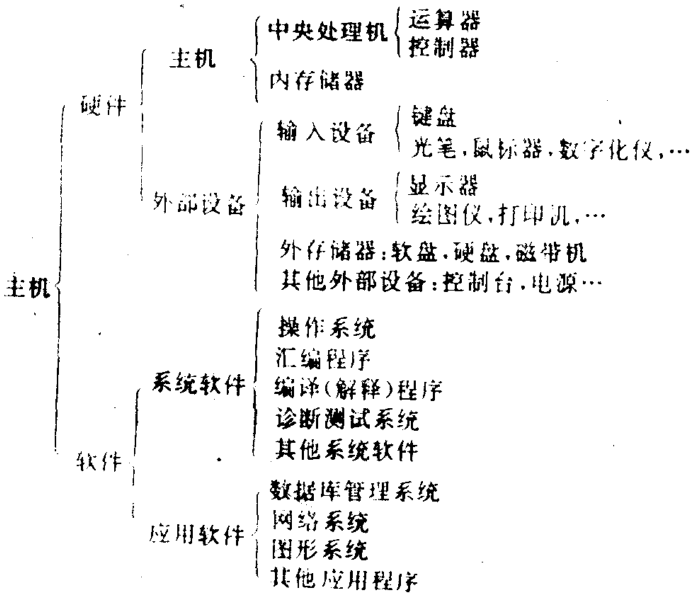

主机
硬件
主机
中央处理机
运算器
控制器
内存储器
外部设备
输入设备
键盘
光笔，鼠标器，数字化仪，…
输出设备
显示器
绘图仪，打印机，…
外存储器：软盘，硬盘，磁带机
其他外部设备：控制台，电源…
软件
系统软件
操作系统
汇编程序
编译（解释）程序
诊断测试系统
其他系统软件
应用软件
数据库管理系统
网络系统
图形系统
其他应用程序

图1.5 计算机系统组成

1.5 计算机中采用的计数制

计算机是采用二进制(Binary Number System)代码来表示信息的。但是在计算机语言中

还采用八进制（Octal Number System）、十进制（Decimal Number System）、十六进制（Hexadecimal Number System）。我们对这方面的知识作一介绍。

1.5.1 数的表示方法

（1）进位计数制。按进位的方法进行计数称为进位计数制。日常生活中一般用十进位计数制。

一个十进制数的数值是由10个不同的数码0，1，2，…，9来表示的。数码所处的位置不同，代表的数的大小也不同。例如：1234，读为一千两百三十四，或者说从右向左起第一位是个位，第二位是十位…。这个“个，十，百，千，…”，在数学上叫“位权”。显然，每一位上的数码与该“位权”的乘积表示该位数值的大小。位权是10^0，10^1，10^2，…。

一种计数制中，数码符号状态的个数称为该计数制的“基数”。那么，十进位计数制的基数是10，它是“逢十进一”的。

十六进制数由0，1，2，…，9，A，B，C，D，E，F共16个数码来表示。十六进位计数制的基数是16，它是“逢十六进一”的。

（2）进位计数制中数的表示方法

在十进位计数制中，15067.2可以写成：

$$15067.2=1\times10^4+5\times10^3+0\times10^2+6\times10^1+7\times10^0+2\times10^{-1}$$

在二进位计数制中，1010.11可以表示为：

$$1010.11=1\times2^3+0\times2^2+1\times2^1+0\times2^0+1\times2^{-1}+1\times2^{-2}$$

一般说来，任何一个十进制数N：

$$N=\pm K_nK_{n-1}\cdots K_1K_0K_{-1}K_{-2}\cdots K_{-m}$$

可表示为：

$$\begin{aligned}N&=\pm(K_n\times10^n+K_{n-1}\times10^{n-1}+\cdots+K_1\times10^1+K_0\times10^0+K_{-1}\times10^{-1}\\&\quad+K_{-2}\times10^{-2}+\cdots+K_{-m}\times10^{-m})\\&=\pm\sum_{i=-m}^{n}(K_i\times10^i)\end{aligned}$$

其中，m，n为正整数，K_i可以是0，1，2，…，8，9共10个数码中的任何一个。十进制数的基数就是10。

表1.1 不同进位计数制数的表示

进位制	基数	位权	特点	通式
十	10	10^i	逢十进一	$\pm\sum_{i=-m}^{n}K_i\times10^i$
二	2	2^i	逢二进一	$\pm\sum_{i=-m}^{n}K_i\times2^i$
八	8	8^i	逢八进一	$\pm\sum_{i=-m}^{n}K_i\times8^i$
十六	16	16^i	逢十六进一	$\pm\sum_{i=-m}^{n}K_i\times16^i$
X	x	X^i	逢X进一	$\pm\sum_{i=-m}^{n}K_i\times X^i$

对于任意进位计数制，基数可以用正整数X来表示。这时数N可表示为：

$$N=\pm\sum_{i=-m}^{n}K_iX^i$$

其中，K_i 是 0，1，…，$(X-1)$中的任何一个。显然，这里基数是 X，“逢 X 进一”，位权是 X^i。

我们用表 1.1 和表 1.2 小结这些内容。

表 1.2　不同进位计数制数码转换简表

十进制	二进制	八进制	十六进制
0	0000	0	0
1	0001	1	1
2	0010	2	2
3	0011	3	3
4	0100	4	4
5	0101	5	5
6	0110	6	6
7	0111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

1.5.2　计算机用二进制表示数

计算机采用二进制来表示数字和用二进制数进行运算，它的优点如下：

(1) 数的表示状态简单。二进制只有 0，1 两种状态，容易表示，容易获得。如脉冲的“有”、“无”，电位的“高”、“低”，晶体管的“导通”、“截止”，电灯的“亮”、“灭”。在这种简单的状态下，工作可靠，抗干扰能力强。

(2) 算术运算简单。四则运算如果是十进制数，必须分别记住 10 个整数的 55 个“和”与 55 个“积”的规律，而二进制的 2 个整数“和”与“积”的规律极为简单，各只有 3 个。

加法：$0+0=0$　　　　乘法：$0\times0=0$

$0+1=1+0=1$　　　　$0\times1=1\times0=0$

$1+1=10$　　　　$1\times1=1$

显然，这样的运算规律比十进制简单得多。

(3) 逻辑线路易于设计。二进制只有两种状态，从而可以应用逻辑代数这一数学工具来对计算机的逻辑线路进行分析和综合设计。

(4) 采用二进制可以节省设备。如果设备状态为某种进位计数制的数码个数和所表示的数的数位的乘积，那么，用十进制表示零，一，二，…，九这 10 个数，需要 0，1，2，…，9 这 10 个数码表示的一位数，这时所需的设备状态为 $10\times1=10$(个)。

又因为$(9)_{10}=(1001)_2$，所以用二进制表示时要 4 位，而二进位只有 0，1 两个数码，则此时的设备状态为 $2\times4=8$(个)。然而，这 8 个设备状态所能表示数的范围可达 0000～1111 即 0～15。

由此可见，二进制比较节省设备。

用十进制表示 0～99 这 100 个数需要多少设备状态？这时需要 0～9 这 10 个数码组成 2 位数，因此，需要 20 个设备状态。因为$(99)_{10}=(1100011)_2$，所以用二进制表示这 100 个数需 0，1 这 2 个数码组成的 7 位数，这时要有 14 个设备状态。并且$(1111111)_2=(127)_{10}$，实际上 7 位二进制数可以表示到十进制数 127。

再利用 $\lg 2=0.3010$，

$$2^N=10^{\lg 2^N}=10^{N\lg 2}\approx 10^{0.3N}$$

上式表示一个 N 位二进制数相当于 $0.3N$ 位十进制数。N 位 2 进制数需要 $2N$ 个设备状态；$0.3N$ 位十进制数需要 $10\times 0.3N=3N$ 个设备状态。这也说明了采用二进制比采用十进制节省存储设备量。

另外，采用二进制可以节省设备，也可以用数学方法来证明，这里不再赘述。

1.5.3 不同计数制间的转换

不同计数制之间的转换根据是：如果两个有理数相等，则两个数的整数部分和小数部分一定分别相等。

(1) 十进制数转换成二进制数

① 十进制整数转换成二进制整数。下面通过具体例子说明转换规律，如把$(215)_{10}$化为二进制数的形式。由二进制数的定义

$$(215)=K_n\times 2^n+K_{n-1}\times 2^{n-1}+\cdots+K_2\times 2^2+K_1\times 2^1+K_0\times 2^0$$

来确定 $K_n, K_{n-1}\cdots K_2, K_1, K_0$ 的值。由上式得：

$$215=2\times(K_n\times 2^{n-1}+K_{n-1}\times 2^{n-2}+\cdots+K_2\times 2^1+K_1\times 2^0)+K_0$$

此式表明 K_0 为 215/2 的余数，$K_0=1$。

上式又可写成：

$$\frac{215}{2}-\frac{1}{2}=2\times(K_n\times 2^{n-2}+K_{n-1}\times 2^{n-3}+K_2\times 2^0)+K_1$$

同理得到 $K_1=1$。

依此类推，可以求出 $K_2, K_3\cdots K_{n-1}, K_n$ 的值。

除 式		余 数
2	215	$K_0=1$
2	107	$K_1=1$
2	53	$K_2=1$
2	26	$K_3=0$
2	13	$K_4=1$
2	6	$K_5=0$
2	3	$K_6=1$
2	1	$K_7=1$

所以，$(215)_{10}=(11010111)_2$。

规则：十进制整数转换为二进制整数时，采用除 2 取余倒排法。

注意：从低位向高位逐次进行，二进制数形式为：$K_nK_{n-1}\cdots K_2K_1K_0$；一直除到商出现 0 为止。

② 十进制小数转换成二进制小数。例如：

$$(0.625)_{10} = K_{-1} \times 2^{-1} + K_{-2} \times 2^{-2} + K_{-3} \times 2^{-3} + \cdots + K_{-m} \times 2^{-m}$$

上式两边都乘以 2，得到：

$$(1.250)_{10} = K_{-1} + (K_{-2} \times 2^{-1} + K_{-3} \times 2^{-2} + \cdots + K_{-m} \times 2^{-m+1})$$

上式右边括号内的部分小于 1，即相当于左边的 0.25 部分，显然 $K_{-1}=1$。又得到：

$$(0.250)_{10} = K_{-2} \times 2^{-1} + K_{-3} \times 2^{-2} + \cdots + K_{-m} \times 2^{-m+1}$$

两边再乘 2，得到：

$$(0.50)_{10} = K_{-2} + K_{-3} \times 2^{-1} + \cdots + K_{-m} \times 2^{-m+2}$$

左边整数部分为 0，所以 $K_{-2}=0$。

继续下去，可以求出：$K_{-3}, K_{-4}, K_{-5}, \cdots K_{-m}$。所以 $(0.625)_{10}=(0.101)_2$

规则：十进制小数转换为二进制小数时，采用乘 2 取整顺排法。

注意：从高位向低位逐次进行，二进制小数为：$0.K_{-1}K_{-2}\cdots K_{-m}$；十进制小数不一定都能转换成完全相等的二进制小数，最后取近似值，直到满足精度要求为止。

③ 十进制数转换成二进制数

将整数部分和小数部分按上述方法分别进行转换，然后再将其组合起来。例如：

$$(215.625)_{10}=(11010111.101)_2$$

(2) 二进制数转换成十进制数

二进制数的每位数码和位权相乘之和就是十进制数。例如：

$$(11010111.101)_2 = 1\times 2^7 + 1\times 2^6 + 0\times 2^5 + 1\times 2^4 + 0\times 2^3 + 1\times 2^2 + 1\times 2^1 + 1\times 2^0 + 1\times 2^{-1} + 0\times 2^{-2} + 1\times 2^{-3} = (215.625)_{10}$$

(3) 二进制数和八进制数之间的转换

二进制数转换成八进制数的规则为：将每 3 位二进制数用 1 位八进制数表示即可；当二进制数带有小数时，以小数点为准分界，3 位一组，分别进行转换，不足 3 位补 0。反之，八进制数转换成二进制数是将每 1 位八进制数用 3 位二进制数来表示并以小数点分界。

例如：

001	001	101	111	.	010	001	011	100	二进制数
|	|	|	|		|	|	|	|	
1	1	5	7	.	2	1	3	4	八进制数

(4) 二进制数和十六进制数之间的转换

由于 $2^4=16$ 关系，并仿照二进制数、八进制数转换方法可以得到：

0001	0110	1001	.	0100	1010	二进制数
|	|	|		|	|	
1	6	9	.	4	A	十六进制数

(5) 八进制数和十进制数之间的转换

请读者从下面两个例子中得出它们的转换方法。

例 1.1　$(105)_8 = 1\times 8^2 + 0\times 8^1 + 5\times 8^0 = (69)_{10}$

例 1.2　把十进制数 128 化为八进制数。因

$$\begin{array}{r|l l}
8 & \underline{128} & (0 \\
8 & \underline{16} & (0 \\
 & 2 & (2
\end{array}$$

故 $(128)_{10}=(200)_8$

(6) 八进制数转换成十六进制数

可以先将八进制数化为二进制数，然后再将二进制数化为十六进制数。

1.5.4 ASCII 码

ASCII(American Standard Code for Information Interchange)码是美国标准信息码。ASCII 码用 7 位二进制编码法，故可表示 128 个字符。在计算机中通常以 8 位二进制作为一个存储单元，即一个字节，所以可以用一个字节表示一个 ASCII 码字符。例如，数字 0 的代码是 00110000，1 的代码是 00110001，字母 A 的代码是 01000001，B 的代码是 01000010 等，它们对应的十进制数分别为 48，49，65，66 等。

1.6 用计算机解决实际问题的过程

应用计算机解决实际问题，首先要分析问题，列出已知条件和目标要求，明确题意；然后进行数学处理，建立数学模型，分析算法，选择最简单并能满足计算精度要求的算法。确定算法是程序设计中的重要环节，它对有效地发挥计算机性能、并得到正确的计算结果将起到重大的作用。

在程序编写前，一般可画出流程图。流程图能把确定的算法用框图描述出来，一个框表示一个运算步骤或过程。图 1.6 是流程图中常用的一些图形符号。

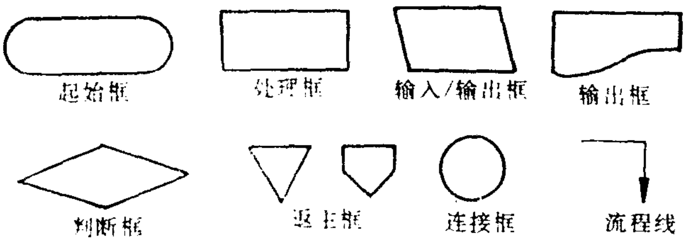


图 1.6 程序流程图符号

根据程序流程图，并按计算机能够接受的方式，选择一种高级语言编写出源程序。源程序输入计算机并经调试运行，直至程序运行可靠，运算结果经分析正确无误，最后才能验收程序，资料整理归档。

用计算机解决实际问题的具体步骤如图 1.7 所示。

一个复杂问题，先从总体开始设计；再逐步精细化地进行分析；最后采用结构化程序设计理论指导程序设计。有关这方面的知识我们将在第九章中进行详细讨论。

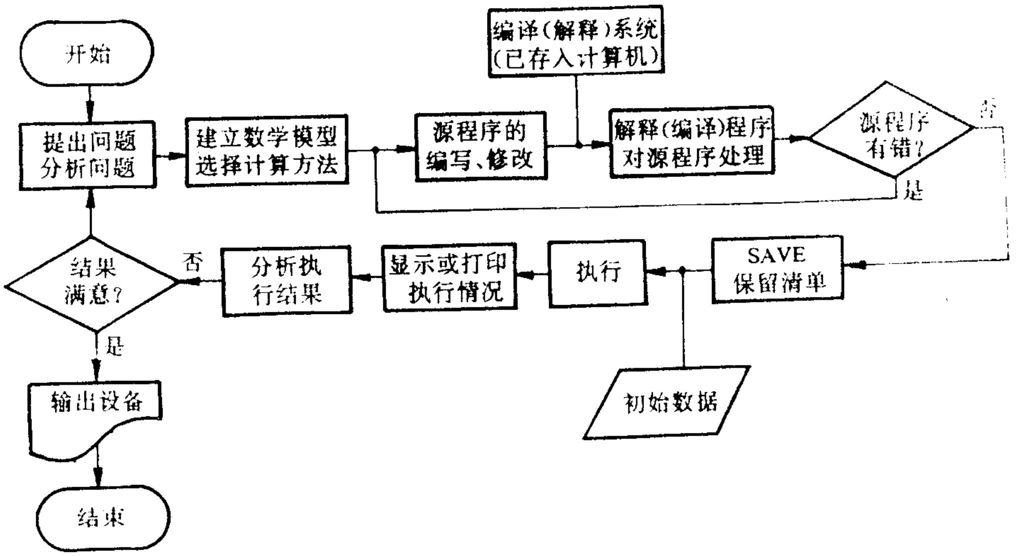

开始
提出问题
分析问题
建立数学模型
选择计算方法
编译(解释)系统
(已存入计算机)
源程序的
编写、修改
解释(编译)程序
对源程序处理
源程序
有错?
否
是
SAVE
保留清单
执行
初始数据
显示或打印
执行情况
分析执
行结果
结果
满意?
否
是
输出设备
结束

图 1.7 计算机解决问题的全过程

小 结

人类不同时期制造和利用了不同工具，不断地为自己的生存和生活同大自然作斗争。直到 1946 年研制成功第一台电子计算机，使人类进入了计算机时代，为人类文明开创了新纪元。

本章着重讨论了计算机的发展和用途，计算机硬件结构和组成，计算机中采用的计数制；最后用流程图的方法表示了计算机解决实际问题的全过程。

习 题

1. 试述人类利用工具的不同阶段，从而说明人类的进步和文明。

2. 从 1946 年第一台 ENIAC 计算机诞生到今天，计算机不断发展体现在哪些方面？

3. 试用具体实例说明当今计算机的用途。

4. 阐述计算机硬件和软件的概念，并说明它们之间的关系。

5. 试说明下列名词的含义：

机器语言　汇编语言　汇编程序
源程序　目标程序　高级语言
解释方式　编译方式　操作系统

6. 计算机中为什么要采用二进制数？试述不同进位计数制中数的表示方法。

7. 完成下列数字转换：

$(128.618)_{10}=(\quad)_2$
$(32767)_{10}=(\quad)_2=(\quad)_{16}$
$(3774)_8=(\quad)_{10}=(\quad)_2$
$(FFFF)_{16}=(\quad)_{10}=(\quad)_2$
$(1010.0101)_2=(\quad)_{10}$
$(10011100.0101)_2=(\quad)_8$
$(3476.521)_8=(\quad)_2$

8. 试述用计算机解决实际问题的全过程。

第2章　微型计算机基本操作

自1946年第一台电子计算机ENIAC运行成功以来，计算机主要朝着大型和微型方向发展。如果说大型计算机在国防建设、尖端科学中起到了巨大作用的话，那么微型计算机(Microcomputer，简称微型机)利用其体积小、使用方便、可靠等优点迅速占领了很多技术领域，为计算机应用和普及谱写了新的篇章。

因为IBM-PC(Personal Computer)是目前国内外广泛使用的典型的微型机，而其它微型机一般都和它兼容，所以本章以IBM-PC微型机为基础，重点介绍微型机的一些基本知识和基本操作。

2.1　微型计算机的发展

60年代末，大规模集成电路制造技术和电子计算机技术都得到很大发展，特别是超大规模集成电路的出现，直接导致了微处理器(Microprocessor)的制造和发展。70年代以来，微型机一经问世，便立即得到社会的承认和青睐，它以诸多吸引人的优点异军突起，其发展日新月异。

2.1.1　微型机硬件的发展

在硬件方面，微型机的主流芯片INTEL系列一直在不停地更新换代。INTEL公司1971年推出第一代4位微处理器4004；1972年到1976年相继推出了第二代8位微处理器8008、8080、8085；1978年到1981年第三代16位微处理器8086、8088以及80186、80188、80286相继出现；1988年以后第四代32位微处理器出现了，这就是当前人们普遍使用的80386、80486等。

由于微型机使用的芯片不断更新，导致了微型机的发展。以IBM-PC为例，它经历了IBM-PC/XT、IBM-PC/AT，AT又分为286AT、386AT以及486AT等。IBM-PC/AT机的容量不断扩大，内存由64KB扩展到256KB，直到1MB或者更大；外存储器已从软盘发展到硬盘，而且容量越来越大，可达到数百MB以上。

利用超大规模集成电路技术，把微型机的CPU(控制器和运算器)集成在一个芯片上，这就是微处理器。同样，利用大规模集成电路技术可以做成容量相当大的内存储器，即随机存储器RAM(Random Access Memory)和只读存储器ROM(Read Only Memory)。人们还常常见到单片机和单板机这类名称，实际上就是把微型机的CPU、一定容量的RAM和ROM以及输入/输出接口电路集成在一个芯片上，成为单片机(Single Chip Computer)；或者把它们集成在一块印刷电路板(称为系统板)上，成为单板机(Single Board Computer)。IBM-PC及其兼容机的CPU、存储器以及输入/输出接口电路都集中在一块系统板和其它选件板上。从IBM-PC机的系统组成来看，系统板是必须的，而选件板是供用户根据需要购置的。将选件板插入系统板的适当槽口内，选件板将与系统板构成一个整体进行工作。

一台微型机硬件，主要包括主机(CPU、内存储器)、磁盘驱动器、显示器(CRT)、键盘以

及打印机等。这些组成如图 2.1 所示。

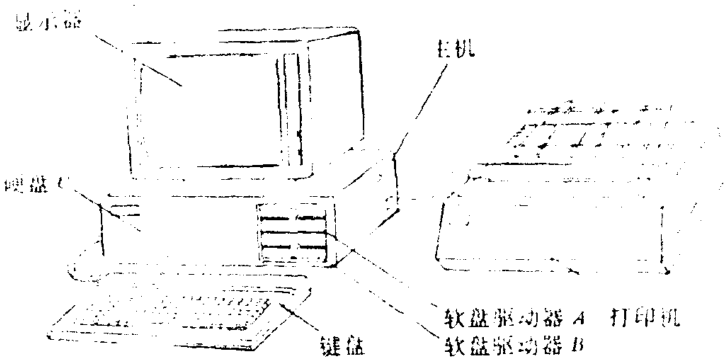


图 2.1 微型机系统组成

2.1.2 微型机软件的发展

1973 年第一台微型机出现以后，微型机的操作系统也得到迅速发展。CP/M(Control Program/Monitor)是 Digital Research 软件公司研制的在 8 位微型机上使用的第一个操作系统，以后相继又有 CDOS(Cromemco Disk Operating System)、MDOS(Motorola Disk Operating System)以及 MP/M 等操作系统问世。

IBM-PC 微型机出现以后，应用了 Microsoft 公司研制的 MS-DOS 操作系统。MS-DOS 版本不断更新，由 MS-DOS1.0 单任务单用户发展到今天的 4.0、5.0 等功能较强的操作系统。MS-DOS4.0 和 5.0 可以支持 Microsoft Net Works 网络系统软件，可以使多个用户联网。CC-DOS(Chinese Character-Disk Operating System)是 MS-DOS 经过汉化和扩充的中文 DOS 系统，它使 IBM-PC 及其兼容机具有汉字输入/输出和处理中文信息功能。另外，当前在不少高档微型机和工作站上已运行了 UNIX 或 XENIX 操作系统，UNIX 和 XENIX 是一个多用户、多任务、分时操作系统。

微型机其它软件的发展更为迅速。在微型机使用的初期主要用汇编语言和 BASIC 语言，现在几乎所有的通用语言都可以在微型机上运行了。微型机上还可以为用户配置丰富多样的应用软件，如数据库管理软件 dBASE、FOXBASE、ORACLE，图形软件 Auto CAD、GKS (Graphical Kernel System)，文字处理系统 WS(Wordstar)、WPS(Word Processing System)、高级科技文献书刊编排系统 4S(Super Science Setting System)等。各类不同用户还根据需要开发了许多专用软件，使微型机的功能更加完善，用途更加广泛。微型机的运算速度也越来越快，可以连接各种输入/输出设备，多台微型机可以联网通信。显然，这些微型机具有过去大型机、中型机的功能和速度，因此人们称之为大微机(Large Microcomputer)。

2.1.3 大微机与工作站

计算机工作站(Workstation)是前几年发展起来的一种大微机，它的主要特点是：

(1) 具有更大容量的内存和外存。通常内存容量由 4MB 到几百 MB，外存容量由几百 MB 到数千 MB。

(2) 采用 32 位 CPU，主流芯片是 MOTOROLA 的 68000 系列。

(3) 大多采用 UNIX 分时、多任务、多用户操作系统。

(4) 具有高分辨率的图形终端和友善的用户界面，分辨率可达 1286×1024。

(5) 大多采用 X-Window 窗口系统，屏幕上可以有多个窗口同时运行程序，一个窗口相

当于一台微型机。

（6）网络环境良好。如 SUN 工作站发展的 SUN NFS(Network File Sestem)现在实际上已成为网络工业标准。

2.2 操作系统和 DOS 的组成与结构

在计算机系统中，一个程序和数据如何送入计算机，内存如何分配，I/O 设备何时启动又如何启动，出了故障如何处理，用户如何和计算机通信，结果信息又如何输出等等，所有这些工作都是由操作系统来管理和控制的。

2.2.1 操作系统概念

操作系统是计算机系统中极为重要的系统软件，它是在硬件（裸机）的基础上利用计算机自身的处理能力设计出来的一整套系统程序。计算机有了操作系统之后，用户可以通过它来控制和使用计算机，因此操作系统是用户与计算机之间的一个接口。图 2.2 展示了操作系统与硬件、其它系统软件以及用户软件之间的关系。

操作系统主要完成下列任务：

（1）处理机管理。在现代计算机中常常有多道作业同时运行，但处理机只有一个，为了提高中央处理机的利用率，操作系统要根据一定的原则来调度处理机。

（2）存储管理。存储管理是对内存资源进行合理分配和管理。多道程序设计的引入要让多个作业共享内存，保证它们之间互不侵犯；还要解决内存容量不足问题，即把一个实际存储器扩充为比本身容量大得多的虚拟存储系统。

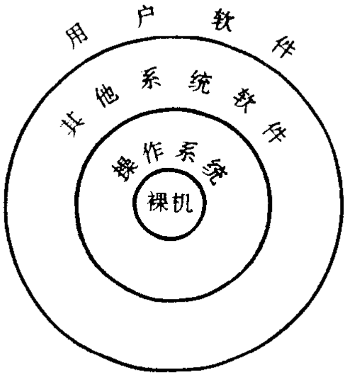


图 2.2 操作系统与硬件、软件之间的关系

（3）设备管理。设备管理的任务是按用户请求进行设备分配，并在完成 I/O 操作后及时回收资源。

（4）文件管理。操作系统把程序、数据、各种信息，甚至外部设备都当作文件来进行统一管理，它负责文件的存取和文件库的管理等。

（5）作业管理和控制。计算机进行操作的一个独立任务叫做一个作业。作业管理包括作业的输入/输出、作业调度和作业控制。

总之，一台计算机系统经过操作系统的一系列改造和扩充之后，可以成为内存容量充分大、外部设备足够多、使用方便、功能齐全的虚拟计算机系统。

根据不同的用途和使用方式，操作系统又可分为几种类型，如单用户单任务操作系统，多道作业批处理系统，分时系统和实时系统，计算机网络操作系统等。不同种类的计算机可以安装不同的操作系统，即使是同一台计算机系统也可以配置几个不同的操作系统。一台 386 微型机可以同时配置 DOS 系统和 XENIX 系统，但是在机器一次启动和关闭周期内只能

运行一种操作系统。

2.2.2 MS-DOS 的组成和结构

起初，MS-DOS 是一个单任务单用户的微型机磁盘操作系统。它的主要功能是对文件进行管理和对外部设备进行管理和控制，如文件的建立、存储、读/写、修改和删除等，以及显示器、键盘、打印机、通信控制器的启动与控制等。

MS-DOS 的组成包括下列四个程序：引导程序，输入/输出设备管理程序 IBMBIO. COM，文件管理和功能调用处理程序 IBMDOS. COM，命令处理程序 COMMAND. COM。下面对这四个程序进行简要的分析。

(1) 引导程序。是一个小程序，它存放在 DOS 系统软盘的 0 面 0 道 1 扇区。每次启动 DOS 时，它能自动装入内存并执行。它先检查系统目录，找到 IBMBIO. COM 和 IBMDOS. COM 的存放位置，并把它们装入内存的指定位置，再将控制转移到 MS-DOS 的启动点，启动 MS-DOS 系统。

(2) IBMBIO. COM。它是 MS-DOS 与 ROM BIOS 的接口模块，是一组 I/O 设备管理程序。BIOS 是一个常驻 ROM 的基本 I/O 系统，用于微型机和外部设备之间的数据传输，处理键盘、显示器、通信设置的信息，处理打印机出错、通信适配器工作不正常等出错信息。

(3) IBMDOS. COM。它是 DOS 系统的核心部分，主要功能是文件处理、磁盘读/写以及外部设备管理等。IBMDOS. COM 是一个负责进行文件管理和功能调用的处理程序，提供系统与用户的高级接口，一般供系统程序员使用。磁盘驱动器信息的存取，磁盘数据的编排组织，每张磁盘上文件目录的记载，文件分配表指明新文件应该存放的地点等都是由 IBMDOS. COM 程序来负责处理的。

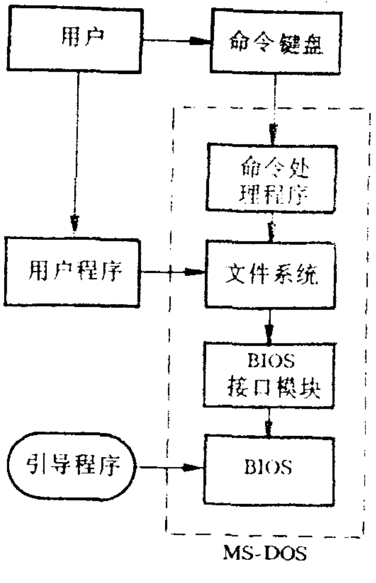


图 2.3 MS-DOS 的组成层次关系

(4) COMMAND. COM。这是一个键盘命令处理系统，它是计算机系统与用户的真正接口，负责分析、解释用户打入的各种命令，并转到相应的程序去执行。其主要功能是：①进行批文件处理，当系统启动时，如果有 AUTOEXEC. BAT 文件，将自动执行；②进行设备中断处理和关键性错误处理；③负责内部命令的解释与处理，外部命令的装入与执行。

IBMBIO. COM 和 IBMDOS. COM 这两个文件目录不显示，COMMAND. COM 是可以直接检查和修改的文件。DOS 启动后上述 4 个程序常驻内存，并为用户提供服务，使用户感到操作起来十分方便。

引导程序、IBMBIO. COM、IBMDOS. COM 以及 COMMAND. COM 之间的结构关系是按图 2.3 所示。

目前，Microsoft 公司已研制出了多个 MS-DOS 版本。新版本不断扩充前面版本的功能，并且新旧版本总是向上兼容的。

2.3　磁盘与磁盘驱动器

磁盘和磁盘驱动器是微型机的重要组成部分。磁盘是微型机上被广泛使用的一种随机存取的外存储器，通常分为软磁盘和硬磁盘两大类。磁盘驱动器是以旋转的磁盘组作为记录媒体的数字磁记录装置。

2.3.1　软磁盘

软磁盘的盘基是采用聚脂塑料制成的，盘面上涂覆约 3μm 厚的磁性材料，用来作为存储数据的载体。软磁盘放在一个永久性硬质保护套内，它的外表结构如图 2.4 所示。图中中心孔是驱动器轴孔，软磁盘装入驱动器后由主轴夹住并带动旋转。长方形孔为磁头读/写槽，露出磁盘片的位置供磁头读/写数据用。索引孔表示磁盘扇区的起始位置，每转一圈产生一个索引信号。写保护缺口位置若封起来表示禁止对磁盘进行写操作，这时只能读出信息而不能再往上面写了。

微型机上常用的软磁盘主要有 5.25 英寸(133mm)和 3.5 英寸(89mm)两种。对于 5.25 英寸软盘初始格式化时，DOS 3.2 把软磁片划分成许多不同半径的同心圆，这些圆形轨道称为磁道。磁盘的每个面分为 40 个磁道，编号从 0～39，最外边是 0 道，最里边是 39 道，信息就记录在磁道上。为了便于读/写信息，把磁道划分为若干个区，这些区呈扇面形，所以称为扇区。DOS 把磁道分为 9 个扇区，每个扇区上可记录 512 个字节(Byte)信息，如图 2.5 所示。

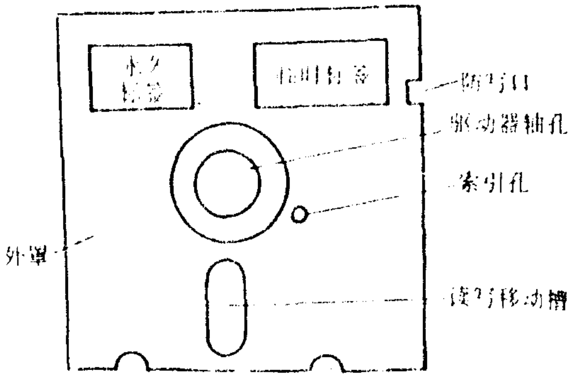


图 2.4　5.25 英寸软盘外形结构

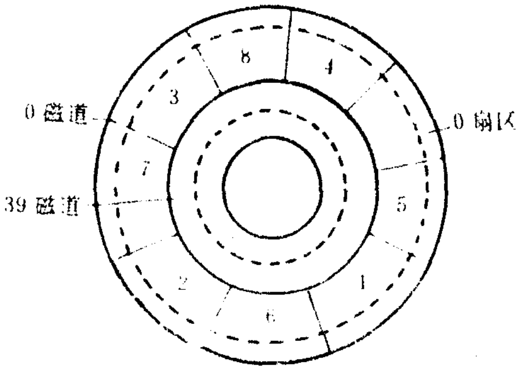


图 2.5　磁盘的分区与分道图

根据上述格式化的方法可知，单面软盘能存储 184320 个字节，双面软盘能存储 368640 个字节。高密度 5.25 英寸软盘容量可达 1.2MB，3.5 英寸软盘容量可达 1.44MB。另外，还有 8 英寸软磁盘，不过现在已经用得很少了。作为外存储器的磁带、磁盘，由于它存储容量很大，所以人们常常称它们为海量存储器。

在计算机中把 8 位二进制称为 1 个字节(1 Byte)，1024 个字节为 1 KB，1 MB 是 1024×1024(字节)＝1048576(字节)。这些常用的存储容量单位应该记住。

2.3.2　硬磁盘

硬磁盘是采用金属基底、其表面涂覆磁性材料的信息存储装置。硬磁盘和软磁盘有很多相同之处，但也有不少区别。例如，软盘驱动器只能用一张盘片，而硬盘驱动器可带动一个盘

组（几个金属盘片组成），它是全封闭、不可更换的固定盘；硬磁盘是刚性材料组成的，转速可达每秒 3600 转，而软盘只能达到每秒 300 转；硬盘运转时，磁头是浮在磁盘表面进行读/写的，而软盘需磁头直接接触磁盘表面来读/写信息。硬盘是一种可靠、高速、大容量外存储器，又称为 Winchester 磁盘。

IBM-PC/XT 和 IBM-PC/AT 以及兼容机可以配置容量达几十 MB 到几百 MB 或更大的硬磁盘。下面给出 IBM-PC 的 10MB 硬磁盘的主要技术指标。

盘尺寸：5.25 英寸

记录面：4

磁道数：306×4=1224

每道扇区数：17

扇区容量：512B

总存储容量：10.4MB

数据传输速率：5Mbit/s

硬磁盘使用之前必须进行初级格式化（一般出厂前做好），进行分区和高级格式化处理。初级格式化工作是对新磁盘划分磁道和扇区，或者由于某种原因对一个已经使用过的硬盘重新划分磁道和扇区。IBM-PC/AT 286 以上机器，可以配置 DOS 系统和 XENIX 系统，这时要对硬盘进行分区。硬盘空间一部分分配给 DOS 使用，另一部分分配给 XENIX 系统。当然，系统中如果只装一个操作系统，硬盘也可以不分区。硬盘的高级格式化是通过 FORMAT 命令来完成的，主要是检查硬盘的格式化、磁道好坏及其替换、清除目录区和文件分配表的内容等。

2.3.3 磁盘驱动器

软盘驱动器由三部分组成，即盘片驱动部分、磁头定位部分和数据读/写抹除部分。盘片驱动部分由直流伺服马达带动盘片作 300 转/分的恒速旋转，当转速稳定后，即开始读/写/抹除工作。读/写/抹除盘片上信息是由相应的电路完成的。磁头定位是采用一步进马达与磁头取数臂连接，步进马达带动磁头取数臂作平面直线运动，寻找磁道和磁道上扇区信息位置。

不同的微型机可以配置一个或两个软盘驱动器（盘标符为 A:和 B:），另外还可以配置 1 个或 2 个硬盘驱动器（盘标符为 C:和 D:）。用户拉开软盘驱动器前面板上的门闩，插入软盘，再关上门闩，驱动器轴能自动定位盘片并将它夹在轴上。驱动器转动时指示灯亮，表明软盘开始工作。硬盘驱动器可以用两个或多个不可卸的盘片来存储信息。对于 5.25 英寸盘片，每个盘片上有上、下两个盘面，每个盘面有 306 个磁道，每道 17 个扇区，每个扇区存放 512 个字节。如果是 4 个盘片，总容量可达到 20MB。

2.4 文件系统概念

存放于磁盘上的一组有关信息的集合称为文件，它可以是一个程序，一组字符串或一组数据信息。每个文件都有自已唯一的名字。磁盘上所有信息都是按文件形式存放的，操作系统是按文件名对文件进行建立、读/写、修改和删除等一系列操作。

2.4.1 文件的命名

一个完整的文件说明，包括驱动器标志、路径、文件名、扩展名等部分。驱动器标志是指存放这个文件的磁盘现在在那个驱动器内，系统缺省驱动器可以是A，也可以用A>B：改变缺省驱动器标志。文件名由1～8个字符组成，这些字符可以是：(1) 英文字母A～Z；(2) 数字0～9；(3) 特殊字符$ # & ! % @ ()-等。文件名不允许出现下列字符：* ?\><|。

扩展名由1～3个字符组成。凡是文件名可以使用的字符，扩展名也都可以使用，当然不是每个文件都必须要有扩展名的。文件名和扩展名之间要用一个圆点“.”把它们分开。

一个好的文件名不但要符合DOS的规定，而且要能反映文件的特点，做到“见名知意”。例如，一个用C语言写成的第一个音乐程序，可以用MUSIC1.C的文件名存放于磁盘上。

一个磁盘上可以存放很多文件，单面软盘根目录下可以存放64个文件，双面软盘根目录下可以存放112个文件，而硬磁盘根目录下可以存放的文件数由DOS分区数的大小来决定。

2.4.2 文件的分类

MS-DOS把文件分为若干种类型。例如，一个文件是ASCII组成的，称为ASCII文件；一个文件由可执行二进制代码组成，称为可执行二进制代码文件等等。磁盘上文件很多，为了区分文件，便于对文件管理和查询，MS-DOS对扩充名的一些约定也表示了文件的分类，如：

COM——系统程序文件
EXE——可执行程序文件
BAT——可执行批处理文件
SYS——系统文件
BAS——BASIC源程序文件
FOR——FORTRAN源程序文件
C——C语言源程序文件
ASM——汇编语言源程序文件
OBJ——目标码文件
BAK——后备文件
TXT——文本文件
LIB——库文件
TRU——True BASIC源程序文件
TRC——True BASIC编译文件
…

在MS-DOS系统中还有一些与外部设备有关的特殊文件，这些文件称之为设备文件。最常用的设备文件名有：

CON：——键盘输入/显示器输出文件
LPTn：PRN：——第n台并行打印机文件
COMn：——第n个异步通信适配器接口文件
NUL：——不存在的虚拟设备，True BASIC允许使用“NUL”作为虚拟设备

把外部设备当作特殊的文件处理，这些文件可与磁盘文件统一起来，以便于它们都由操作系统进行管理。

2.4.3 文件目录

为了对大量文件进行管理，MS-DOS 系统采用了目录结构，其形式类似于一棵倒挂的树，所以又称为倒树型目录结构如图 2.6 所示。

操作系统为每个软盘和硬盘都建立了一个目录结构，这个结构的最上层是根目录，通常用反斜线\表示，这个目录在每个盘上只有一个，是在系统格式化时产生的。在根目录下可以建立其他目录或文件，产生的目录为根目录的子目录，而根目录是这些子目录的父目录或称之为上一级目录。在图 2.6 中 PAY.DBF 是根目录下的文件。总之，每个目录都可以挂接多个子目录和文件。一个系统中可以有多少级目录和存放多少个文件，原则上没有限制，主要由磁盘的容量和磁盘驱动器以及格式化的方法决定。

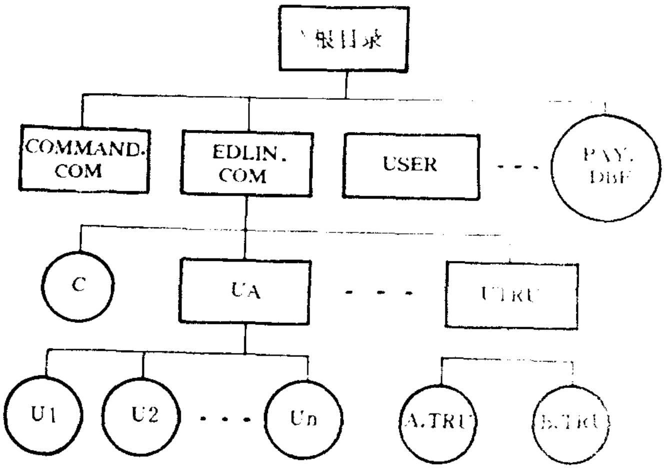


图 2.6 MS-DOS 目录文件结构图

MS-DOS 在每个磁盘中都开辟了两个区域，一个是目录区，另一个是文件分配表(FAT)区。当用 FORMAT 命令格式化磁盘时，首先产生一空的根目录，然后再设置一张 FAT 表用来记录文件在磁盘上的存储位置以及自由空间的位置。MS-DOS 就是根据目录区和文件分配表的内容来管理、检索盘上文件和盘空间区域的。

2.4.4 路径和路径名

在图 2.6 MS-DOS 目录结构中，如果要存取一个文件就要知道这个文件在目录结构中的位置。例如，若要存取 A.TRU 文件，可以按如下顺序找到它的位置：

\USER \UTRU\A. TRU

它表示查找文件从根目录开始经过一定的路径才找到文件的所在位置。象这样由根目录开始，再接反斜线\隔开的子目录，最后紧跟一个文件名的序列称之为“路径名”(PATH NAME)，或称之为文件路径名。这个路径名除文件名外的部分称为“路径”(PATH)，即：

\USER\UTRU\A. TRU
路径
路径名

一个路径名中可包含若干个目录名和一个文件名，但是不得超过 63 个字符。用户可以按照给定的目录名去搜索文件，如果一个路径名以反斜线\开始，这是从目录系统的根目录开始搜索指定文件，这样的路径称为“绝对路径”，如果不从根目录开始搜索文件，而从中间某个目录(当前目录)开始搜索文件，称为“相对路径”，所谓相对路径是指从当前工作目录开始搜索文件的路径。

2.4.5 通配符

MS-DOS 提供了总体文件标识符，又称为通配符。通配符可以标识磁盘上许多不同文件，从而可以提高 DOS 对文件处理的效率。

“*”表示几个不确定的字符。例如：*.C 指当前磁盘上所有具有扩展名为.C 的文件；EXMP.* 指所有具有不同扩展名的 EXMP 文件；*.* 指当前目录下的全部文件；“?”仅表示一个不确定的字符，例如：?KU.BAT 可以表示以下多个文件标识符：

AKU.BAT
BKU.BAT
CKU.BAT
1KU.BAT
…

关于通配符的使用要注意两点：一是通配符的作用仅限于当前工作目录上的所有文件；二是通配符仅能使用在文件名和扩展名的位置上，而不能用于目录中。

2.5 常用 DOS 命令

用户使用 MS-DOS 命令，可以进行磁盘格式化，软磁盘的复制、比较，还可以进行文件的复制、比较、输出、改名和删除以及目录结构管理等等。下面我们对 DOS 常用命令分类进行简要介绍。

2.5.1 外部命令

外部命令的处理程序是作为文件驻留在磁盘上，只有键入命令名和所需要的参数时，它才被读入内存并执行。这就是说使用一个外部命令时，当前盘上必须有这个外部命令，否则 DOS 将会告知你这条命令没有找到。

(1) FORMAT 命令

格式：FORMAT [d:][/s][/1][/4][/8][/v][/b]

用途：① 格式化磁盘，把磁盘设置成 DOS 所能接受的记录格式。② 寻找并标出磁盘上坏的磁道和扇区。③ 建立目录和文件分配表(FAT)，使磁盘可以接收 DOS 文件。

说明：① d：格式化磁盘所在的驱动器号。② /s 使 DOS 的系统文件复制在磁盘上，格式化的磁盘可以用来引导系统。③ /1 格式化一个单面软盘；/4 用高密驱动器格式化双密度磁盘；/8 磁盘格式化后，8 扇区/磁道，否则为 9 扇区/磁道。④ /v 磁盘格式化时加上卷标号。⑤ /b 格式化后，8 扇区/磁道，留出存放 DOS 系统的空间(此时可以存放各种版本的 DOS 系统，否则只能存放 DOS2.0)。

例 2.1 A>FORMAT B:

显示：

```
Insert new diskette for drive B:
    and strike ENTER when ready
```

按 ENTER 键，显示：

```
Formatting...
```

又显示：

```
Formatting...format complete
```

```
...
Format another ？(Y/N) 根据需要回答Y或N。
```

例 2.2　把B盘格式化为带卷标的系统盘，可使用如下命令：

```
A>FORMAT B：/s/v
```

注意：① 格式化将破坏原来盘上所有信息，不能对写保护的盘进行格式化。② /s 系统信息在0道上，因此除了0道外，允许磁盘有坏的磁道，坏的磁道被保留不用。

(2) DISKCOPY 命令

格式：DISKCOPY [d：][d：][/1]

用途：把源磁盘上内容原样拷贝到目的盘上，称之为“全盘拷贝”。

说明：① 格式中第一、二参数分别是源驱动器号和目的驱动器号。② /1 仅拷贝软盘的第一面。

注意：① 如果目的盘没有格式化，在全盘拷贝时进行格式化，显示：Formatting while copying。② DOSKCOPY 只适用于软盘间的拷贝，不适用于软盘和硬盘之间的拷贝。③ 此命令清除目的盘上原有信息。

例 2.3　A>DISKCOPY A：B：

显示：

```
Insert Source diskette in drive A：
Insert Target diskette in drive B：
Press and key when ready…，按键后
Copying 40 tracks
9 Sectors /Track，2 Side(s)
Copy complete
Diskcopy another (Y/N)?
```

最后回答Y或N，表示有或没有软盘继续拷贝。

(3) DISKCOMP 命令

格式：DISKCOMP [d：][d：][/1][/8]

用途：比较两个指定磁盘内容是否一致，一般做完 DISKCOPY 命令后，应当运行 DISKCOMP 命令。

说明：参数的用法基本同 DISKCOPY 命令。

打入比较命令后，显示：

```
...
Comparing 40 tracks
9 Sectors per track，2 side(s)
Compare OK
Compare another diskette(Y/N)? 最后回答Y或N。
```

注意：DISKCOMP 命令将参与比较的磁盘以磁道为单位逐个进行比较；若比较两个或两批文件则用 COMP 命令，它是以字节来进行比较的。

(4) SYS 命令

格式：SYS d：

用途：把缺省驱动器DOS盘上的 IBMBIO.COM 和 IBMDOS.COM 两个系统文件复制

到指定驱动器的软盘上。这里只需指定目的驱动器磁盘号。

例 2.4　A>SYS B:

把两个系统文件从 A 复制到 B。若要用 B 盘来引导系统，必须再用 COPY 命令把 COMMAND.COM 内容拷贝到 B 盘中。

(5) CHKDSK 命令

格式:CHKDSK[d:][filename][/F][/V]

用途:分析磁盘文件目录和分配表，并产生磁盘和内存状况的报告。

说明:① 指定文件名时，将显示它占用磁盘的情况。② /F 用来查找文件分配表和目录中的错误并自动修正。③ /V 显示磁盘上所有文件及其路径。

例 2.5　A>CHKDSK B:

显示:Volume abc1 created (日期) (时间)

```
...bytes total disk space
...bytes in ... hidden files
...bytes in bad sectors
...bytes available on disk
...bytes total memery
...bytes free
```

外部命令还有:MODE、COMP、LABEL、BACKUP、RESTORE、PROMPT、TREE、MORE、FIND、SORT 等，其中有些将在目录命令中介绍。

2.5.2 **内部命令**

内部命令包含在 COMMAND.COM 中，它随着 DOS 的启动装入内存并常驻内存，所以当用户键入这些命令时能立即执行。

(1) CLS 命令

格式: CLS

用途: 清除显示屏幕。

(2) DIR 命令

格式: DIR [d:][path][filename][.ext][/p][/w]

用途: 列出所有的文件目录或指定的文件目录。

说明:① 在 DIR 命令后选择文件名和扩展名，就可以列出指定的文件目录。② /p 当文件目录较多时，一屏显示后暂停，按任一键后继续显示。③ /w 横向显示目录，每行可显示 5 个文件目录，此时不显示文件的大小、建立日期和时间。

例 2.6　A>DIR

显示:Volume in drive A in Mydisk

```
Directory of A:
File1 A 10333 7-20-90 11:12p
...
LEVEL2 <DIR> 7-07-91 12:10p
5File(s) ... bytes free
```

例 2.7　A>DIR LINK.EXE

显示：LINK EXE 39896 5—09—92 12:12p

(3) COPY 命令

因为 COPY 命令的格式比较复杂，所以我们不去追究命令格式。这里先说明 COPY 命令的用途，然后通过例子来重点学习 COPY 命令的用法。

用途：① 复制一个或多个文件，并可改变它们所属目录和文件名。② 用 COPY 命令可以合并文件。③ 可以在系统设备间传送数据。

例 2.8 A>COPY B:EXMP.EXE

将指定驱动器 B 上文件 EXMP.EXE 拷贝到 A 盘当前目录中，文件名不变。

例 2.9 A>COPY *.* B:

将 A 盘当前目录中的文件全部拷贝到指定盘 B 上，文件名不变。

例 2.10 A>COPY TT1.BAS TT2.BAS

在原盘上拷贝，所属目录不变，但改变文件名。

例 2.11 A>COPY TT1.BAS B:TT2.BAS

文件从当前盘拷贝到 B 盘，改变文件名。

例 2.12 A>COPY A1.C+A2.C+B:B1.C AAB.C

将 A1.C、A2.C 和 B 盘上的 B1.C 合并为 AAB.C 新文件。

例 2.13 A>COPY CON: FILE1

通过键盘输入建立文件 FILE1。当文件内容输入完毕时按 CTRL—Z 或 F6 键作为结束标志。

```
A>COPY CON: LPT1
```

通过键盘输入到第一台打印机上。

```
A>COPY TEST.PAS CON:
```

文件 TEST.PAS 送 CRT 显示。

```
A>COPY TEST.PAS PRN:
```

文件 TEST.PAS 打印输出。

注意：① 例 2.13 都是在系统设备间进行信息传送。② CON：作为源文件时表示键盘，作为目的文件时表示显示器。③ COPY 命令是以文件为基础的，一个文件一个文件地拷贝。

(4) TYPE 命令

格式：TYPE [d:][path][filename][.ext]

用途：在 CRT 上显示指定文件的内容。

说明：如果在显示文件的同时想打印文件，则在发此命令前同时按 CTRL—P 键，让打印机连机，若再按一次 CTRL—P 则表示脱机。

例 2.14 A>TYPE B:TT.HLP

显示(或同时打印)TT.HLP 文件的内容。

注意：TYPE 命令只能显示正文文件，不能显示其它文件。

(5) REN (RENAME)命令

格式：REN [d:][path] filename [.ext] filename[.ext]

用途：把第一个 filename[.ext]名字改为第二个 filename[.ext]名字。

例 2.15 C>REN B:ABC.BAS A1.TRU

把 B 盘上文件 ABC.BAS 改名为 A1.TRU。

例 2.16　A>REN B:ABC1 *.TRU

把 B 盘上文件 ABC1 改名为 ABC1.TRU。

(6) DEL 命令

格式：DEL [d:][path][filename[.ext]]

用途：从指定或缺省驱动器中删除指定文件。

说明：① DEL 和 ERASE 等效。② DEL 命令不能删除隐含文件，也不能删除. 和.. 子目录标志文件。

例 2.17　A>DEL B:TT1.TRU

删除 B 盘上文件 TT1.TRU

例 2.18　A>DEL B:TT.*

删除文件名为 TT 的所有文件，扩展名为通配符 *。

例 2.19　DEL *.*

Are you sure [y/n]?

系统提示用户是否确实需要删除所有文件，根据实际情况回答 Y 或 N。

内部命令还有 DATE、TIME、VER、VOL、MD、CD、RD、PATH、REM、PAUSE、ECHO、GOTO、IF DOS 以及 SHIFT 等等，这些命令有些在目录命令中介绍，有些可以通过上机来学习。

2.5.3 目录操作命令

通过前面的学习，我们已经知道 MS-DOS 系统是一个倒树型目录结构，它是通过目录类命令来进行管理的。目录类命令可以帮助用户建立、删除、显示和管理目录结构。

(1) MD(MKDIR)命令

格式：MD [d:] path

用途：在指定或缺省驱动器的磁盘目录上建立子目录。

例 2.20　C>MD\TTBB

这是在 C 盘根目录下建立一个名叫 TTBB 的子目录。

例 2.21　C>MD LEVEL

在 C 盘当前目录下建立 LEVEL 子目录。

(2) RD (RMDIR)命令

格式：RD [d:] path

用途：删除指定驱动器磁盘上或当前工作目录中的一个子目录。

例 2.22　A>RD C:\TTBB\LEVEL

删除 C 盘上 LEVEL 子目录。

例 2.23　A>RD \AA\BB1

删除当前盘上 BB1 子目录。

注意：在要删除的目录中，不能存在任何文件和子目录。

(3) CD (CHDIR)命令

格式：CD [d:] path

用途：改变当前目录或显示当前目录路径。

例 2.24　C>CD

```
C>\   显示当前工作目录是根目录
C>CD \TTBB\LEVEL   改变当前工作目录
C>CD
C>\TTBB\LEVEL   显示当前工作目录，显然工作目录已经改变了
C>
```

(4) TREE 命令

格式：TREE [d:][/F]

用途：显示所有的目录路径，如用/F 选择项还可以显示子目录上的文件。

下面的两条命令，读者可上机一试：

① C>TREE|MORE（显示目录路径）。

② C>TREE/F|MORE（显示目录路径和文件）。

说明：MORE 的用法是：(输入源)|MORE，这是将输出送到显示器(或打印机)上，一屏一屏地显示输出，按任一键后继续显示下一屏。

2.5.4 批处理命令

将常用的一些命令放在扩展名为.BAT 的文件中，这个文件叫批处理命令文件。批处理命令文件具有以下特征：它的扩展名一定是.BAT；它包括一个或多个 DOS 命令；批处理命令中可以带参数；命令可以“嵌套”。另外，AUTOEXEC.BAT 是一个特殊的批处理命令文件。当启动 DOS 时，COMMAND.COM 程序会自动查找 DOS 所在的磁盘根目录中有无 AUTOEXEC.BAT 文件，如有该文件，则 DOS 会自动执行 AUTOEXEC.BAT 文件中的各个命令；如没有该文件，则 DOS 提示用户输入日期和时间，从此起动了机器。

建立批处理命令的方法，一是用行编辑命令 EDLIN，直接建立一个批处理命令文件；二是用 COPY 从键盘上输入。下面通过例子来介绍用 COPY 建立批处理文件的方法。

例 2.25　C>COPY CON:AAAA.BAT

```
CLS              清屏
DATA              日期
TIME              时间
VER               显示 DOS 当前版本号
CTRL—Z (F6)         输入结束
```

建立了一个 AAAA.BAT 批处理文件。

运行 AAAA.BAT：

```
C>AAAA
Current date is Mon 11—10—1992
Enter new date (mm—dd—yy):
C>TIME
Current time 19:36:20.12
Enter new time:
C>VER
IBM Personal Computer DOS Version 3.20
```

```
C>
```

这样一旦加电启动 DOS 或热启动时，系统将自动清屏，显示日期、时间，最后显示 DOS 的版本号。

例 2.26　建立一个含有参数的批处理文件 COPYFILE. BAT，适用于复制、比较以及显示文件所占的字节数。

```
A>COPY CON: COPYFILE. BAT
CLS
COPY %1 %2
COMP %1 %2
DIR %2
F6        输入结束
```

当执行 COPYFILE. BAT 可更换参数的批处理文件时，形式参数%1、%2 被源文件名和目的文件名所取代。运行如下命令：

```
A>COPYFILE A:MODE. COM B:MODE1. COM
```

这里%1 被 A:MODE. COM 取代，%2 被 B:MODE1. COM 取代。事实上，执行 COPYFILE. BAT 文件效果等价于：

```
CLS
COPY A:MODE. COM B:MODE1. COM
COMP A:MODE. COM B:MODE1. COM
DIR B:MODE1. COM
```

DOS 可支持 10 个批处理参数，它们是%0～%9。

为了加强批处理的功能，DOS 系统提供了 7 条子命令，它们是：

① REM [remark]——显示批命令间的注释内容。

② PAUSE [remark]——暂停系统批处理操作。

③ ECHO [on/off/remark]——控制显示批处理命令名称 。

④ FOR % % varible IN (set) DO DOS-Command——允许 DOS 命令反复被执行。

⑤ GOTO [lable]——把控制转向到指定标号的命令行上去执行。

⑥ IF [NOT]Condition DOS-Command——根据条件处理命令。

⑦ SHIFT——左移批处理参数，允许使用多于 10 个的可代换参数。

上述 7 个子命令均为内部命令，具体格式和使用方法可参考有关 DOS 手册。

小　结

本章以 IBM-PC 微型计算机为基础，重点介绍了微型计算机硬件和软件的发展，操作系统的简单概念，MS-DOS 系统的组成和结构，磁盘和磁盘驱动器，文件的分类，倒树型文件目录以及 MS-DOS 常用命令。DOS 常用命令从存放方式上可分为外部命令和内部命令，从使用方式上又可分为文件管理命令、目录管理命令、设备管理命令以及批处理命令。这些常用 DOS 命令，我们可以通过上机练习，逐步熟练地掌握它们。

习　题

1. 试述微型计算机硬件和软件的发展。什么叫大微机？什么叫工作站？

2. 试述操作系统和计算机硬件、软件的关系。操作系统在计算机中的作用是什么？为什么微型机要用 MS-DOS 来启动？

3. MS-DOS 是由哪几个部分组成的？试描述这几部分的层次关系。

4. 磁盘为什么要进行格式化？试述磁盘格式化后的结构和容量。

5. 为什么人们常称外存储器磁盘、磁带为海量存储器？

6. 文件如何命名？文件如何分类？详细叙述倒树型文件目录结构。

7. 举例说明通配符的作用。

8. MS-DOS 常用命令从存放方式上如何分类？从使用方式上又如何分类？

9. 试上机执行 10 个内部命令和外部命令，从而了解这些命令的用法。

10. 什么是批处理命令？举例说明建立批处理命令的方法。

第 3 章　数据与数据操作

在程序设计语言中，数据是计算机程序处理的对象，每个数据都属于某个特定的数据类型，不同类型的数据的表示方法、存储形式及运算规则都不相同。True BASIC 中数据类型分为基本类型和构造类型，构造类型的数据是由基本类型数据按一定规则构造得到。本章介绍基本类型的数据及其操作，这是 True BASIC 语言和程序设计的基础。

3.1　常量与变量

True BASIC 基本类型有数值和字符串两类，每一类型的数据又可分为常量和变量，本节介绍 True BASIC 基本类型的常量及变量的表示方法和语法规则。

3.1.1　常量

常量是指在程序执行过程中固定不变的原形数据，常量有数常量和字符串常量两类。

(1) 数常量

数常量又称常数，其大小范围，不同的机器有不同的限定。True BASIC 语言中数的表示方法有以下两种：

①一般表示法。这种表示法的形式为：

[<数符>]<整数>[<. 小数>]

正数的数符可任选，可以省略，也可以冠以正号(+)数符，负数则必须冠以负号(－)数符。对整数，小数部分可省略；小数部分以小数点(.)开始。整数或小数部分包含 1 位或多位十进制数字。在 True BASIC 中，数的最多有效位是 10 位(在一些计算机中可能会更多)。当数值有效位超过 10 位时，计算机用四舍五入的近似值表示。在常数中，除小数点外，不允许插入其它任何符号(如空格，逗号等)。

②科学表示法。又称指数表示法，用来表示非常大的数或非常小的数的方法，由数字部分和指数部分组成，其形式如下：

[<数符>]<整数>[<. 小数>]E/e[<指数符>]<指数>

由于在计算机中所有的符号都是一个一个地在同一水平行上输入和输出，每个符号占据一格，无法表示指数部分右上角的幂次，因此，指数部分由字母“E”或“e”开始，表示以 10 为底的指数，后跟指数符(+/－)，正指数的指数符同样可以省略，指数是 1 位或 2 位十进制整数，表示 10 的幂。

注意：数的科学表法不允许省略数字部分。例如 10000 可以表示成 1E4，不能写成 E4，否则，计算机将把它当作一个数值型变量来处理，其初值为 0，详见本节变量部分。表 3.1 是合法表示和不合法表示的数的例子。

数可以有任意位，但 True BASIC 规定正数的取值范围是 10^{-99}～10^{99}，最大有效位是 10 位。当数值超过最大值 10^{99} 或小于最小值 -10^{99} 时，则出现“OVERFLOW”的溢出错误；当数值大于 -10^{-99} 或小于 10^{-99}，非常接近于 0 时，则以“0”代替，称机器零。

表 3.1

合法	不合法	不合法原因
+34 或 34	34+	数符必须冠于数之首
-100000000	-10,000,000	不允许插入逗号
-12.345	3.14 159	不允许插入空格符
1E3 或 1e3	E3 或 e3	不允许省略数字
3.14E-15	8/3	数值表达式

不同机器的数具有不同的取值范围，True BASIC 提供了两个函数，可以用来查询所用机器的极限值：函数 MAXNUM 返回该机器所能表示的最大正数，EPS(0)函数返回该机器所能表示的最小正数。

(2) 字符串常量

在计算机应用中，经常会遇到许多非数值的数据，如某个人的姓名、通信地址、工作单位等，一般把这些非数值的数据称为字符串，它是 True BASIC 语言另一基本类型的数据。所谓字符串常量是指包含在两个双引号间的一串字符和符号，简称字符串，这些字符和符号是计算机键盘上规定使用的字符集中的成员(附录 1)。例如，“BEIJING”，“SHANHAI”，“A12”，“12.50”等都是字符串常量。

双引号是用来确定字符串常量起止的界符，计算机原样输出双引号中的字符而忽略双引号。双引号也可以作为字符串中的字符，但此时必须重复两次双引号。

例如，“”“True”“BASIC”是合法的字符串常量，输出结果为“True” BASIC。

字符串中包含的字符的个数(不包含双引号)称为字符串的长度。True BASIC 允许字符串最大长度是 32767 个字符。由空格符组成的字符串称为空格串，它的长度是空格符的个数；只有界符，不包含任何字符的字符串称为空串，其长度为 0。

在字符串常量中，英文字母的大小写是有区别的。例如：“TRUE”和“true”是两个不同的字符串。常数和数字字符串是两种不同类型的数据，例如 123 和“123”，123 是常数，可以用于算术运算，而“123”是字符串，不能用于算术运算。

3.1.2 变量

变量是指在程序执行过程中可以改变的数据。True BASIC 中的变量可分为简单变量、下标变量和数组变量。简单变量存储基本类型数据，本节只介绍简单变量。

为了便于识别，在程序设计语言中，变量都用一些符号来命名，这些符号称为变量名。True BASIC 对变量名的命名有以下规定：

(1) 变量名必须以英文字母开头，其后的字符可以是英文字母，数字和下划线“_”组成。

(2) 一个变量名最多允许由 31 个字符组成。

(3) 英文字母的大小写等效。

例如：A，AB，C12，a_5，xyz 等都是合法的变量名，而 A+B，3AB，“abc”或 A.5，AB%等则是不合法的变量名。AB 和 ab 则是同名变量。

与常量相对应，变量按其中所存储的数据的类型不同可分为数值变量和字符串变量两种。数值变量只能存储数值常数，字符串变量只能存储字符串，这两种变量名的命名规则相同，只是字符串变量名必须以"＄"字符作后缀，以区别数值变量，字符“＄”也算在 31 个字符

之内。True BASIC 自动给未定义的变量赋初值：数值变量初值为 0，字符串变量初值为空串。例如，AB＄，B＄，c2＄等均为合法的字符串变量名。

在程序设计语言中，变量的含义不同于与数学中的变量，它实质上是变量名命名的存储单元中的数据，是一个确定的、已知的数据，并在程序执行过程中可以根据需要改变其中所存储的数据，显然，在程序执行的某一时刻，它是一个已知数据。程序中的变量，都是按变量名存取数据，任何一个变量参与运算时，总是取它所存储的具体数据来进行，程序执行期间可以多次从变量中读出数据，或根据需要改变变量中所存储的数据，写入新的数据。一旦写入新的数据，则变量中原存储的数据便不再存在，而以新数据替代。因此，变量具有不破坏读出、破坏性写入的特性。

3.1.3 保留字

在 True BASIC 中有些标识符是作为保留字。保留字是留作系统本身使用的标识符，不允许用作变量名。True BASIC 语言的保留字包括 6 个关键字，不带自变量的函数名和 3 个内部子程序名，详见表 3.2。

表 3.2 True BASIC 的保留字

关键字	函数名		子程序名
ELSE	DATE	MAXNUM	DIVIDE
ELSE IF	DATE＄	PI	PACKB
IF	DET	RND	POCK
NOT	EXLINE＄	TIME	
PRINT	EXTEX＄	TIME＄	
REM	EXTYPE		

除了表 3.2 中作为保留字的函数名外，虽然可以使用 True BASIC 中其它的函数名作为变量名，但为了避免混乱，建议不要使用语句关键字、标准函数名及子程序名作为变量。

3.2 数值运算与数值表达式

True BASIC 的表达式是指用符合规定的运算符和括号将常量、变量或函数连接起来的数据处理式。根据数据操作运算所得结果的数据类型不同，True BASIC 有四种表达式：即数值表达式、字符串表达式、关系表达式和逻辑表达式。本节和下节分别介绍数值表达式和字符串表达式。

3.2.1 数值表达式

数值表达式是用 True BASIC 规定的算术运算符和括号将常数、数值变量或数值处理函数连接起来的计算式。数值表达式的运算结果为常数。数和数值变量是最简单的数值表达式。

3.2.2 算术运算符

True BASIC 规定允许在数值表达式中使用的算术运算符有以下 5 种：

运算符：+，－，*，/， ^

含　义：加，减，乘，除，乘方

数值表达式中的算术运算符与一般计算公式中的书写不同。应注意以下几点：

(1) 在数值表达式中乘号“＊”不能省略，也不能用“×”或“·”代替。例如，A和B两变量相乘，应写成 A＊B，而不能写成AB或A×B或A·B。

(2) 除号和乘方运算符只能用“/”和“^”表示。例如，A除以B表示成A/B，A的B次方表示成A^B，这是因为在计算机中输入和输出时，表达式中所有符号都在同一水平行上，每个符号占据一列，而无上下标之分。

(3) 在表达式中允许使用圆括号或方括号，不允许使用花括号，圆括号和方括号等效，且括号必须成对出现。

(4) 在表达式中不允许连续使用两个运算符，如需要时，可用括号加以分隔。例如表达式A＋(－B)或－B＋A是合法的表达式，而A＋－B则是不合法的表达式。

例如代数式 0.627H－0.6654W－42.7946 的True BASIC的表达式为：

0.627＊H－0.6654＊W－42.7946

3.2.3 数值处理标准函数

数值处理函数可以完成特定的数值运算。在程序设计中，经常会遇到某些常用函数的计算，如果在使用这些函数时，都由使用者来编写函数的处理程序，那将是一件十分繁琐的工作。为了使用方便，True BASIC提供了一些常用的函数和子程序，供用户调用，这些函数就称为标准函数。标准函数是由语言本身提供的程序进行处理，用户在使用时，可以直接在表达式中调用标准函数或用调用语句调用子程序。

系统提供的标准函数的格式为：

函数名〔(＜形式参数表＞)〕

其中，函数名是函数的标识符，形式参数表中的参数是函数的自变量，有的标准函数可以没有形式参数。

数值处理函数可以在数值表达式中直接调用，其调用方法为：

函数名〔(＜实在参数表＞)〕

其中〔〕号表示其中的内容为可选项，说明有的函数不需要参数，如PI，RND和MAXNUM等，调用时只需写出函数名即可；有的函数则需要有一个或多个形式参数，如SIN(X)，SQR(X)，MAX(X，Y)和MIN(X，Y)等，调用时必须给出相应的实参数，且应用圆括号括起来，对多形参函数、各实参之间还须用逗号予以分隔。数值处理函数的实参可以是常数、数值变量、数值表达式或是数值处理函数。

标准子程序DIVIDE(X，Y，q，r)有4个形式参数，其中X，Y是值参数，用作给子程序输入计算值；q，r是变量参数，用作向调用程序回送子程序的计算结果。子程序用CALL语句调用，其形式如下：

CALL DIVIDE(＜实参数表＞)

其中，实参数表括在圆括号中，实参数的个数、安排顺序应与形式参数一一对应，并用逗号分隔，其中对应值参数可以是数值，也可以是已经定义的数值变量、数值处理函数或数值表达式，而对应变量形式参数的参数必定是变量，可以与形式参数同名，亦可以用不同的变量。该程序被调用时，从调用程序接收被除数送x，除数送y，完成x/y计算后，商取整后送q，余数送r，返回调用程序。

数值处理函数和子程序的含义、应用举例见表3.3。

表 3.3　数值处理标准函数和子程序

函数名	含　义	应用举例	结　果
PI	返回的 π 值	PI	3.14159
SIN(X)	求 X 的正弦值,X 为弧度,取值为 0～2π	SIN(PI/4)	0.707
COS(X)	求 X 的余弦值,X 为弧度,取值为 0～2π	COS(PI/4)	0.707
TAN(X)	求 X 的正切值,x 为弧度 X≠(n+1)π/2,n 为任意整数	TAN(PI/4)	1
ATN(X)	求 X 的反正切值,ATN(X)的值为弧度, 函数值为－π/2～π/2,－π<x<π	ATN(0)	0
		ATN(PI)	1.26263
DEG(X)	将 X 的弧度值变换为角度值	DEG(PI/4)	45
		DEG(PI)	180
RAD(X)	将 X 的角度值变换为弧度值	RAD(180)	3.14159
		RAD(45)	0.785398
ANGLE(X,Y)	求出 X 轴与原点(0,0)和 坐标点(X,Y)所成射线的夹角	ANGLE(1,1)	0.785398
		ANGLE(1,0)	0
		ANGLE(0,1)	1.5708
		ANGLE(－1,0)	3.14159
ABS(X)	求 X 的绝对值$\lvert X\rvert$	ABS(5)	5
		ABS(－3)	3
SQR(X)	求 X 的平方根$\sqrt{X}$,X≥0	SQR(4)	2
		SQR(9)	3
EXP(X)	求 e^x 的值,e=2.71828	EXP(0)	1
		EXP(1)	2.71828
LOG(X)	求以 e 为底的自然对数 lnX,X>0	LOG(10)	2.30259
		LOG(1)	0
LOG2(X)	求以 2 为底的对数 $\log_2 X$,X>0	LOG2(10)	3.32193
		LOG2(4)	2
LOG10(X)	求以 10 为底的常用对数 lgx,x>0	LOG10(10)	1
		LOG10(0.001)	－3
INT(X)	求不大于 X 的最大整数	INT(3.6)	3
		INT(－3.6)	－4
SGN(X)	符号函数		
	$\mathrm{SGN}(X)=\begin{cases}1 & X>0\\ 0 & X=0\\ -1 & X<0\end{cases}$	SGN(5)	1
		SGN(0)	0
		SGN(－5)	－1
RND	产生大于等于 0 而小于 1 的随机数		
ROUND(X)	求 X 经四舍五入的整 数,等价于 INT(X+0.5)	ROUND(4.32)	4
		ROUND(4.61)	5
		ROUND(－8.5)	－8
		ROUND(－8.51)	－9
ROUND(X,N)	求 X 的第 n 位四舍五入的近似值。 n≥0,小数点右边的第 n 位为四舍五入的近似值。 n<0,小数点左边的第 n 位为四舍 五入的近似值,舍弃位以"0"代替。	ROUND(15.325,2)	15.33
		ROUND(15.325,0)	15
		ROUND(15.325,－1)	20
		ROUND(15.325,－2)	0
TRUNCATE(X,N)	截取 X 第 n 位的近似值。n≥0, 截取到小数点右边第 n 位。 n<0,截取到小数点左边第 n 位,被舍弃位以"0"代替。	TRUNCATE(137.29,1)	137.2
		TRUNCATE(137.29,0)	137
		TRUNCATE(137.29,－1)	130
		TRUNCATE(137.29,－2)	100

续表 3.3

函数名	含　义	应用举例	结　果
MAX(X,Y)	求 X,Y 两者中的最大数	MAX(9,7)	9
		MAX(−4,−7)	−4
MIN(X,Y)	求 X,Y 两者中的最小数	MIN(9,7)	7
		MIN(−4,−7)	−7
EPS(X)	求影响 X 精度的最小数	EPS(1e13)	0.031115
		EPS(0)	1.1154e−308
MOD(X,Y)	求 X/Y 的余数,Y≠0,符号与 Y 一致,等价于 X−Y * INT(X/Y)。	MOD(10,2)	0
		MOD(9,5)	4
		MOD(−7,2)	1
		MOD(7,−2)	−1
REMAINDER(X,Y)	求 X/Y 的余数,Y≠0,符号与 X 一致。	REMAINDER(15,2)	1
		REMAINDER(6.3,2)	0.3
		REMAINDER(−1,2)	−1
		REMAINDER(14,2)	0
		REMAINDER(7,−2)	1
MAXNUM	求机器所能表示的最大正数	MAXNUM	3.59539e+308
子程序 DIVIDE(X,Y,q,r)	求 X/Y 的商 q 和余数 r,Y≠0. 必须用 CALL 语句调用。	CALL DIVIDE(17,8,q,r)	q=2,r=1
		CALL DIVIDE(4,20,q,r)	q=0,r=4

说明:

(1) 在系统启动后,未加设置时,三角函数和 ANGLE 函数的角度一律使用弧度单位,为了使用三角函数和计算方便,True BASIC 提供了角度单位选择语句:

OPTION ANGLE DEGREES——语句选择度为角度单位

OPTION ANGLE RADIANS——语句选择弧度为角度单位

(2) 有关四舍五入求近似值的有两个函数:ROUND(X)和 ROUND(X,n)。

ROUND(X,n)函数实现的等效式为:INT(X * 10 ^ (n+1)+0.5)/10 ^ (n+1)。

ROUND(X)是 n=1 时函数 ROUND(X,n)的特殊情况,等效于 INT(X+0.5)。

这两个函数返回的结果在 X>0 的情况下完全正确,但在 X<0 的某些点上将会出现不符合四舍五入原则的错误,我们认为这是系统实现的疏忽,提请注意。例如:ROUND(−8.5)或 ROUND(−8.5,1)返回的结果为−8,而正确结果应该是−9。

3.2.4 数值运算的优先顺序

如何正确书写 True BASIC 的数值表达式是数值计算的关键,表达式的书写必须遵循运算优先序。在数值表达式中,各运算符按其运算优先顺序进行运算,优先级高的先运算,优先级低的后运算。在 True BASIC 表达式中,函数或乘方优先运算,其次为乘或除运算,最后是加或减运算。相同优先级按从左到右的顺序依次进行运算。

如果在表达式中使用了括号,则括号中的表达式最先运算,如有多重括号嵌套,则内层括号中的运算优先于外层括号。True BASIC 的数值运算的优先顺序可表示如下:

括号→函数/乘方→乘/除→加/减

例如表达式:X ^ 2+3 * (5/(3+8)−3)−SQR(X),运算时,最先计算(3+8),再计算 5/(3+8),然后计算(5/(3+8)−3),接着计算 X ^ 2 和函数 SQR(X),最后是乘号“ * ”,加号“+”和减号“−”的运算。它表达了计算式:$X^2+3(\frac{5}{3+8}-3)-\sqrt{X}$。

例如，代数式 $M\frac{R_1R_2}{G_1G_2}$的数值表达式是 M＊R1＊R2/(G1＊G2)，而表达式 M＊R1＊R2/G1＊G2 代表的是代数式 $M\frac{R_1R_2}{G_1}G_2$。由此可见，在表达式中，应适当使用括号，以改变数值运算的优先顺序。

3.3 字符串表达式与字符串操作

字符串表达式是用字符串运算符及圆括号或方括号将字符串常量、字符串变量或字符串处理函数连接起来的字符串处理式，其运算的结果为字符串。字符串常量和字符串变量是最简单的字符串表达式。

在字符串的处理中，对字符串可以进行多种操作，这里只介绍字符串连接、子字符串截取这两种操作。

3.3.1 字符串操作

(1) 字符串连接操作

用字符串连接运算符“&”将两个或多个字符串连接起来而获得一个新的字符串，这种运算就称为字符串的连接。下面是几则字符串连接操作的例子：

“SHANG”&“HAI”	结果为“SHANGHAI”
“BEI”&“JING”	结果为“BEIJING”
“True”&“_”&“BASIC”	结果为“True_BASIC”

(2) 子字符串截取操作

True BASIC 中字符串的长度可多达 32767 个字符，如果只想使用其中一部分连续的字符，那么可以使用子字符串截取操作取出其中所需字符组成一个新的字符串，称为给定字符串的子字符串，简称子串。

子字符串截取操作格式为：

<字符串表达式>(a:b)/[a:b]

它的作用是从给定字符串中截取从第 a 个字符开始到第 b 个字符为止的子字符串。其中 a，b 为数值表达式，其值分别用来确定子串截取的起始位置和终止位置。a，b 的值应为正整数。若为小数，则自动四舍五入后取整，当 a<1 时，自动以 1 代替；当 a>b 时，则产生一个空串；当 b 大于被操作字符串长度时，则以串长度代替。使用<字符串表达式>[a:MAXNUM]则可以得到字符串中从第 a 个字符到串尾的子串。

以下是子串截取的几则例子：

“SHANGHAI”[2:5]	结果为：“HANG”
“SHANGHAI”[−1:2]	结果为：“SH”
“GOOD”(3:2)	结果为空串“”
“GOOD”(1:7)	结果为：“GOOD”
“SHANGHAI”[5/2:9/2]	结果为：“ANG”
“BEIJING”[4:MAXNUM]	结果为：“JING”

3.3.2 字符串操作的优先顺序

在字符串表达式中，字符串操作的优先顺序是字符串处理函数——→子串截取操作——→

字符串连接操作。有关字符串处理函数将在第 4 章中介绍。在字符串表达式中，可以使用圆括号或方括号来控制字符串操作的优先顺序，括号中的操作优先。若有多重括号，内层括号中的操作优先于外层括号的操作。例如：

“WU”&“SHANGHAI”[2:5]　　结果为：“WUHANG”

(“WU”&“SHANGHAI”)[2:5]　　结果为：“USHA”

3.4　变量的赋值与数据输出语句

在利用计算机处理数据时，数据处理的过程总是包括数据输入、数据处理和结果输出。在 True BASIC 语言中，每个过程都是由相应的语句来完成的。所谓语句，是指能完成某种特定功能的 True BASIC 指令，它由三部分组成：<语句符>[<语句体>]<语句结束符>。

语句符又称为关键字，用来命令计算机执行某种操作的指令。语句符是由语言系统规定的，不能自造或修改，否则将出现句法错误。True BASIC 的关键字由英文单词表示，大小写字母等效。语句体是指定该语句的操作内容，有些语句可以没有语句体部分。语句结束符表示语句结束，True BASIC 语言的语句结束符是回车换行符(Enter 键)，每个语句都必须以语句结束符结束。

本节介绍数据输入和数据输出语句。

3.4.1　变量赋值(LET)语句

格式：LET　<变量>[{<，变量>}]=<表达式>

格式中[　]表示其中的内容为可选项，{　}表示其中的内容可以重复。LET 是赋值语句的关键字，不能省略。符号“=”称为赋值号，赋值号左边最少要有一个变量，且只能是变量，右边是表达式，可以是数值表达式或字符串表达式，常量、变量或函数都是最简单的表达式。

功能：执行该语句时，先计算“=”号右边表达式的值，并将结果按从左到右的顺序逐个赋给“=”号左边的各个变量。因此，赋值语句具有计算和赋值两种功能，先计算后赋值。例如：

LET A=3+2，先计算出表达式 3+2 的值 5，再把 5 赋给变量 A。

LET B,C,D=2 * 3，先计算出表达式 2 * 3 的值 6，并依次赋给变量 B,C,D。

LET E$ =“WU”&“SHANGHAI”[2:5]，将“WUHANG”赋给变量 E$。

说明：

(1) 赋值号“=”不同于数学中的等号。在赋值语句中，LET X=X+1 是表示将变量 X 的当前值加上 1 再赋给变量 X，每执行一次，变量 X 值加 1，称计数器，在程序设计中具有十分重要的作用。在数学中，等号表示相等关系，X=X+1 则是不成立的等式。

(2) 一个赋值语句只能完成一个表达式的计算和赋值，因此，赋值号右边只能有一个表达式，而不能有多个表达式。

(3) 赋值号右边表达式的数据类型必须与被赋值的变量的数据类型一致。例如：

LET A=3　　把常数赋给数值变量

LET A$ =“ABC”　　把字符串常量赋给字符串变量

以上两个均是合法的赋值语句，而 LET A=“ABC”或 LET A$ =3 则是不合法的赋值语句，因赋值号两边的数据类型不一致而出错。

(4) 未经赋值的变量，系统自动赋初值，字符串变量初值为空串，数值变量初值为 0。

(5) 一个赋值语句可以将一个表达式的值赋给一个或多个变量，当给多个变量赋值时，各变量之间必须以逗号分隔。

3.4.2 字符串变量修改语句

利用 LET 语句可以给数值变量及字符串变量赋值，同时使用 LET 语句还可以对字符串变量中存储的字符串进行修改。字符串变量修改语句格式为：

LET ＜字符串变量＞(a:b)/[a:b]=＜字符串表达式＞

该语句的作用是将字符串变量中存储的字符串，用赋值号右侧的字符串表达式的值修改后再赋给字符串变量。它可对字符串变量中存储的字符串实施子串插入、替代、字符删除等修改功能。其中 a，b 为数值表达式，a 的值表示字符串变量中字符串修改的开始位置，b 为修改的截止位置。a，b 的值均应为正整数，若为小数，则自动四舍五入取整。当 a＜1 时，则以 1 代替；当 b 大于串长度时，则以串长度代替；当 a＜b 时，将字符串变量中的字符串从第 a 个字符开始的(b－a)＋1 个字符组成的子串由串表达式的值替代；当(b－a)＋1 大于串表达式的值的长度时，将删去替代后的多余字符；当 a＞b 时，则将串表达式的值插入串变量中存储的字符串的第 a 个字符前，且不删除任何字符。以下是字符串变量修改的几则例子：

LET A$="abcdefghi"　　将字符串"abcdefghi"赋给字符串变量 A$。

LET A$[3:5]="123"　　用字符串"123"替代 A$ 中第 3 到第 5 个之间的字符，A$="ab123fghi"(替代)。

LET A$[6:10]="123"　　用"123"替代 A$ 中第 6 到第 9 个之间的字符，并删除多余字符，A$="abcde123"(替代、删除)。

LET A$[5:5]=""　　删除 A$ 中第 5 个字符，A$="abcdfghi"(删除)。

LET A$[7:MAXNUM]=""　　删除 A$ 中第 7 个以后的字符，A$="abcdefg"(删除)。

LET A$[1:0]="123"　　把"123"插在 A$ 前，A$="123abcdefghi"(插入)。

LET A$[3:2]="123"　　把"123"插在 A$ 中第 3 个字符前，A$="ab123cdefghi"(插入)。

3.4.3 数据输出(PRINT)语句

输出语句可以用来在屏幕上显示数据，如果和通道一起使用，则可将数据写入磁盘上的文件中或由打印机打印。

格式：PRINT　[＜表达式＞][{，/；＜表达式＞}]

其中，/代表或的意思，表示取其左边或右边内容之一。

在输出语句中，可以有一个或多个表达式，可以没有表达式。如果有多个表达式时，各表达式之间以逗号或分号分隔，逗号或分号是 PRINT 语句输出项在同一行中列表位置控制符。PRINT 语句最后亦可以用逗号或分号结束。

功能：执行该语句时，先计算出 PRINT 语句中各表达式的值，再按规定的输出位置在屏幕上显示出语句中各表达式的值。因此，输出语句具有计算和输出两种功能，先计算后输出。

输出语句中的表达式可以是数值表达式或字符串表达式，变量、常量或函数都是最简单的表达式。当输出项为常数时，则直接显示输出常数，字符串常量原样输出不包括双引号界符的所有字符；当输出项为变量、函数或表达式时，则显示输出变量、函数或表达式的值，而

不是变量名、函数名或表达式本身。例如：

```
PRINT 3+2         显示输出结果为 5
PRINT "ABC"       显示输出结果为 ABC
```

输出语句的数据列表输出格式：

PRINT 语句在屏幕上显示数据时，允许一行显示一个数据或一行列表显示多个数据，PRINT 语句中的列表位置控制符(,/;)，使得输出的数据有以下几种不同的输出格式：

(1) 按标准域输出格式

True BASIC 语言显示文本时，通常把显示器屏幕划分为 25 行 80 列，一行可显示 80 个字符，每行又等分为 5 个显示区域，每个域的宽度为 16 个字符，称为标准域。各标准域的划分如图 3.1 所示。

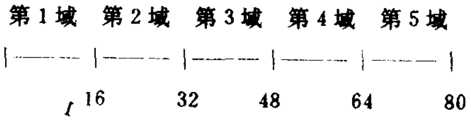


图 3.1

当 PRINT 语句中的各输出项之间以逗号分隔时，则各输出项的值依次在屏幕上的每个标准域的开始处显示，一个标准域显示一个数据，屏幕上每行可显示 5 个数据。当输出项超过 5 项时，则自动换行在下一行输出剩余的数据。如果 PRINT 语句中有两个连续的逗号，则自动跳过一域。这种输出格式称为标准域输出格式。输出数值时，在正数和零前面留有一个空格。例如：

```
OK. PRINT 1,-2,3,-4,5,-6,0,,8
 1        -2        3        -4        5
-6         0                  8
OK.
```

每个标准域可显示 16 个字符，如果数据长度不超过 16 个字符，则在一个域中显示；如果长度超过 16 个字符，则在下一个域继续显示，其后续输出项则顺推一个域显示。例如：

```
OK. PRINT "The People's Republic of China","USA","JAPAN"
The People's Republic of China USA                JAPAN
OK.
```

(2) 紧凑输出格式

当 PRINT 语句中的各输出项之间以分号分隔时，则后一输出项紧跟在前一输出项后输出，这种输出格式称为紧凑输出格式。输出数值时，在正数和零前面留有一个空格，并在输出的数值后面保留有一个空格。例如：

```
OK. PRINT -3;5;-3*5
-3   5   -15
OK.
```

如果两个字符串按紧凑格式输出，则串与串之间无空格地输出。例如：

```
OK. LET A$ ="The People's Republic o"
OK. PRINT A$ ;"f China"
The People's Republic of China
```

```
OK.
```

(3) 混合输出格式

在一个 PRINT 语句中，各输出项之间既有用逗号分隔，又有用分号分隔时，称为混合输出格式。如果输出项前是用逗号分隔，则该输出项在当前域的下一个域输出；如果输出项前是用分号分隔，则该输出项紧接在前一输出项后输出。例如：

```
OK. PRINT “a=”;3,“b=”;5,“T=”;3+5
a=3          b=5          T= 8
OK.
```

3.4.4 单语句立即执行方式

True BASIC 语言系统启动后，整个屏幕划分为编辑窗和背景窗两个文本窗口，并在背景窗口的最下一行显示“OK.”提示符，称为命令行，用以输入系统命令。编辑窗用来输入、编辑 True BASIC 源程序或文本，背景窗用来输入语言系统命令和记录系统运行“历史”背景。

True BASIC 采用解释执行方式，它不仅具有程序执行方式，而且还允许单行语句作为系统命令直接在背景窗命令行输入，并立即执行获得结果，称单语句立即执行方式，适用于语句学习和程序调试。True BASIC 的许多单行语句都可作单语句命令，立即执行。本节介绍的 LET 语句、PRINT 语句皆能用作语句命令，立即执行完成简单计算。例如：计算半径为 10 厘米的圆面积：

```
OK. LET R=10
OK. LET A=PI * R^ 2
OK. PRINT ″AREA(R=10)=″;A;″CM^ 2″
AREA(R=10)= 314.159 CM^ 2
OK.
```

注意：LET 语句立即执行，只将表达式计算值赋给变量存储，显示器没有输出反映。

3.5 True BASIC 源程序

True BASIC 源程序是用 True BASIC 语言对算法的描述文本，是要求计算机完成某种特定任务的语句集合。与立即方式不同，源程序由系统命令 RUN 控制，按语句顺序一批执行。

3.5.1 True BASIC 源程序的构成

True BASIC 源程序由一系列简单语句和控制结构组成，True BASIC 源程序的构成具有一定的规则。

True BASIC 语言规定每个语句行只允许有一个语句，每个语句行的长度可多达 32767 个字符，每一行都必须以回车符结束。

每个简单语句都由一个关键字开始，并且独占一行。关键字的作用是规定该语句的执行功能。控制结构由一个关键字开始，并由另一个关键字结束，其中可包含多个语句，并可占用多个语句行。

每个 True BASIC 程序都必须以 END 语句结束，且 END 语句不得省略，一个程序只能有一个 END 语句。END 语句是程序执行的结束标志，它的作用是使程序停止运行。END 语句的格式为：

```
END
```

为了增加程序的可读性，可以在程序中任何地方使用注释语句注释整个程序标题、作用或一段程序或一个语句的功能。注释语句的格式为：

REM <字符串>

其中 REM 可用“!”代替。

注释语句是非执行语句，即程序执行时，会自动跳过注释语句而不作任何操作，它的作用仅仅是对程序或语句的解释或说明，从而增加程序的可读性。注释语句可以在程序中的任何地方使用，当 REM 或“!”在语句首部时，则表示该行是注释语句，而当在语句后使用注释语句时，则只能使用“!”作为注释的开始，不能用 REM 语句，用以注释其左边的语句功能。

3.5.2 程序设计流程

在应用计算机处理问题时，程序设计是使用计算机的关键。通常，一个复杂问题的程序设计，需要经过以下几步：

(1) 分析问题，确定算法。在编写程序时，首先应对所要处理的问题的需求进行认真分析，以求对问题有一个正确的认识和理解。一旦弄清了问题的性质，就必需确定相应算法，但对同一问题可能有几种不同的算法，因此必须选择一个最恰当的算法，以提高程序的质量。

(2) 根据算法写出计算步骤。计算步骤是对算法的描述，是算法的一种表现形式，它可以使问题变得直观明了。对于简单问题，可以省略此步。

(3) 根据计算步骤编写程序。计算步骤确定后，再使用 True BASIC 语言规定的相应语句来实现算法各步骤的功能，进行程序编写。

(4) 程序调试及运行。在编写好程序并送入计算机后，可在背景窗的命令行打入 RUN 命令，运行程序。如程序没有错误，则计算机就会执行程序，并按要求输出运行结果；如果程序有错误，则显示出相应的错误信息，等待用户作出相应的编辑修改，然后再运行。一个复杂的程序往往不是一次编写，就能通过并得到正确结果，而是需要反复修改、调试，才能保证程序的正确性。

3.5.3 简单程序

简单程序是由一系列简单语句组成的，如赋值语句、输入/输出语句等，它是一种顺序结构的程序。在执行时，各语句是按它们出现的先后顺序执行的，没有程序转移或循环。简单程序只能完成一些简单问题的计算。

下面是一个求解一元二次方程 $2X^2+18X+23=0$ 的两个实根的 True BASIC 简单程序，读者可以从中了解 True BASIC 程序的构成规则及运行结果。

```
! Computing quadratic equation’ s real roots
!
LET A=2
LET B=18
LET C=23
LET X1=(-B+SQR(B^ 2-4*A*C))/(2*A)
LET X2=(-B-SQR(B^ 2-4*A*C))/(2*A)
PRINT "X1=";X1,"X2=";X2
END
........................
OK. RUN
X1=-1.54196        X2=-7.45804
OK.
```

3.5.4 程序执行方式中的 PRINT 语句

在程序中的 PRINT 语句除上节介绍的功能外，还有下列功用。

(1) 当程序中的 PRINT 语句末尾以逗号或分号结束时，那么在该 PRINT 语句输出后光标不换行，后续 PRINT 语句的输出项，则以标准域格式或以紧凑格式在该行继续显示输出项的值。

例 3.1

```
LET a=3
LET b=5
LET c=7
PRINT a,b,c,
LET T=a+b+c
PRINT "T=";T;
LET V=T/3
PRINT "V=";V
END
…………………
OK. RUN
  3     5     7     T=15   V= 5
OK.
```

(2) 换行输出：当程序中的 PRINT 语句末尾没有逗号或分号结束时，则在该 PRINT 语句显示输出后，光标自动换行，后续 PRINT 语句中的各输出项在下一行输出。

例 3.2

```
PRINT 1,2
PRINT 3,4
END
………………
OK. RUN
  1     2
  3     4
OK.
```

(3) 显示空行：在 PRINT 语句中没有语句体时，则显示输出一个空行。而当上一个 PRINT 语句末尾以逗号或分号结束时，则它起取消同行列表位置控制符逗号或分号的作用，光标回车换行。

例 3.3

```
PRINT 1,-2
PRINT
PRINT -3,4,
PRINT
PRINT 5;6;
PRINT
PRINT 7
END
………………
OK. RUN
  1    -2

 -3     4
  5     6
  7
OK.
```

(4) 打印机打印输出：PRINT 语句只能在屏幕上显示数据，如果需要从打印机上打印数据时，则应使用带通道号的 PRINT 语句，并在执行 PRINT 语句前打开打印机通道。打开打印机通道可以使用 OPEN 语句，其格式为：

OPEN #<通道号>:PRINTER

其中，通道号为 1～999 之间的任意整数。

使用 OPEN 语句打开打印机通道号后，再使用语句：

PRINT #<通道号>:<输出项表>

便可将各输出项的值由打印机上打印出来，数据的打印格式与屏幕显示格式相似。

例 3.4

```
OPEN #2:PRINTER
LET R=10
LET S=PI*R^2
PRINT #2:"S=";S
```

```
END                                          OK. RUN
.................
```

执行该程序后，屏幕上不显示结果，程序的运行结果将由打印机上打印出来。

3.6 键盘输入数据语句

赋值语句给变量赋值，数据都是在程序中给定，对于固定数据的程序，每次运行都只能得到同样的结果，因而程序不具有灵活性和通用性。为此，True BASIC 提供了键盘输入数据方式，它是一种人机对话式的输入方式，数据不是在程序中给定，而是在每次运行过程中，用人机对话方式，根据不同需要从键盘上输入不同的数据并获得不同的结果，因此使程序具有灵活性和通用性。

3.6.1 键盘输入(INPUT)语句

格式：INPUT [prompt<提示字符串>:]<变量>[{<,变量>}][,]

其中，prompt<提示字符串>是可选项，如有只能有一个，且只能紧跟在关键字 INPUT 之后，与后面变量之间用冒号分隔，提示字符串必须用双引号括起来，它主要用以提示要赋值的变量名称、类型、数量和顺序等，以免用户输入出错。

INPUT 语句中最少要有一个变量，亦可以有多个变量。当有多个变量时，各变量之间用逗号分隔，并允许在语句中最后一个变量末尾以逗号结束。

功能：使 INPUT 语句中的各个变量依次从键盘上取得数据。

执行 INPUT 语句时，暂停程序的运行，当有 prompt<提示字符串>可选项时，则在屏幕的背景窗口显示 prompt 后的字符串，作为数据输入的提示信息，等待用户从键盘上响应输入，当要求输入多个数据时，输入数据间用逗号分隔，用回车键结束输入，它把输入的数据按顺序依次赋给各个变量。如果没有可选项 prompt<提示字符串>，则在背景窗显示一个“?”号，等待用户从键盘上响应输入数据。

例 3.5

```
INPUT PROMPT"A,B,C=":A,B,C
PRINT A,B,C
END
.................
OK. RUN
A,B,C=1,2,3
 1          2          3
OK.
```

或

```
INPUT A,B,C
PRINT A,B,C
END
.................
OK. RUN
? 1,2,3
 1          2          3
OK.
```

在执行上述程序时，当执行到 INPUT 语句时，暂停程序的运行，并在屏幕的背景窗中显示 A,B,C=或“?”号，用户响应从键盘上输入 1,2,3 三个数据，并以回车符结束输入，则将这三个数据分别赋给变量 A,B,C，然后接着执行其后续语句。

说明：

(1) 在 INPUT 语句中只能出现变量，可以是数值型变量或字符串型变量，且最少要有一个变量，亦可以有多个变量，各变量之间以逗号分隔。

(2) INPUT 语句不能进行运算，因此从键盘上输入的数据必须是不带运算符号的常量，不得是变量、函数或表达式，各数据之间以逗号分隔，且必须以回车符结束。特别要注意的是键入的数不能是分数或带乘方的数，如 1/3 或 10^ 3 均不是常数，而是表达式，而负数(如 -3)则是一个常数。

(3) 当用户响应 INPUT 语句时，从键盘上输入的数据的个数不得少于 INPUT 语句中变量的个数，当输入的数据个数不足时，将显示如下信息，要求继续输入不足的数据，直到满足为止：

Too few input items, Please add more.

当输入的数据个数太多时，则忽略多余的数据，并显示如下信息，但程序正常运行不出错：

Too many input items, Excess ignored.

用户响应输入的数据可以在一行内一次输完并以回车符结束，亦可以分多次进行输入，此时可在输入行的最后加上一个逗号，再打回车，在下一行继续输入数据，直到输完最后一个数据后，直接以回车符结束。值得注意的是，对于任何一个输入错误，不论其后的数据是否正确，都必须重新输入从不正确的输入项开始的所有数据。

例 3.6

```
INPUT A,B,C,D
PRINT A+B+C+D
END
..................
OK. RUN
? 1,2,3
Too few input items, Please add more.
? 4
  10
OK. RUN
```

```
? 1,2,3,4,5,6
Too many input items, Excess ignored.
  10
OK. RUN
? 1,
? 2,
? 3,4
  10
OK.
```

(4) INPUT 语句中各变量的数据类型可以不同，但从键盘上输入的数据与 INPUT 语句中对应的各变量的数据类型则必须一致。当从键盘上给字符串变量赋值时，如果字符串常量的首尾部带有空格或其中含有逗号时，则该字符串必须用双引号括起来，否则可以省略双引号对，即使键入数字亦不会出错，它被当作数字字符串赋给字符串变量；在给数值型变量输入数据时，如果键入数字串(带双引号对中的数字)，则系统自动转换成数值赋给变量，但如果键入的是除数字外的字符，则显示出错信息：

String given instead of number, Please retry from bad items.

说明字符串代替数值输入而产生类型不一致的错误，并要求从不正确的输入项开始重新输入数据。

(5) 在 INPUT 语句末尾以逗号结束时，如果只输入与该 INPUT 语句中变量个数等量的数据，则当执行其后继 INPUT 语句时，显示“Too few input items, Pleaseadd more.”信息，要求继续输入不足的数据；如果输入数据的个数多于该 INPUT 语句中变量的个数，则多余的数据被暂存，并由其后继的 INPUT 语句读入，当多余数据的个数少于其后继 INPUT 语句中变量的个数时，则执行后继 INPUT 语句时，亦显示上面的信息，请求添加不足的数据。对于

prompt<提示字符串>可选项只能在该 INPUT 语句中使用，而在其后继 INPUT 语句中则不能再使用，否则显示“Must be avariable.”出错信息，中止程序的运行。

例 3.7

```
INPUT prompt"num=";a,
PRINT a,b,c,d,e
INPUT b,c,d,e
PRINT a,b,c,d,e
END
………………
OK. RUN
num=8
  8    0    0    0    0
Too few input items, Please add more.
```

```
? 9,1,2,3
  8    9    1    2    3
OK. RUN
num=8,9
  8    0    0    0    0
Too few input items, Please add more.
? 7,6,5
  8    9    7    6    5
OK.
```

由例可见，程序中由于第一个 INPUT 语句是以逗号结尾，它与后继 INPUT 语句的作用相当于一个 INPUT 语句，在第一个 INPUT 语句满足数据要求后，程序继续执行，直到第二个 INPUT 语句被执行时，出现信息请求添加数据。输入的数据个数可以超出第一个 INPUT 语句中的变量的个数，超出者自动赋给下一个 INPUT 语句中的变量。注意，此时多余者只作暂存，并未赋给第二个 INPUT 语句中的变量，所以变量 b 仍被认为未定义，直到第二个 INPUT 语句被执行时，才被定义确认，所以在第一次输出时，变量 b 的值仍为 0。

例 3.8　已知梯形的上底、下底和高分别为 A=5，B=8，H=4，编程求梯形的面积。

```
INPUT prompt"A,B,H=";A,B,H
LET AREA=(A+B)*H/2
PRINT A,B,H,"AREA=";AREA
END
………………
```

```
OK. RUN
A,B,H=5,8,4
  5    8    4     AREA= 26
OK.
```

3.6.2　字符串行键盘输入(LINE INPUT)语句

格式：LINE INPUT [prompt<提示字符串>:]<字符串变量>[{<,字符串变量>}]

功能：当执行该语句时，暂停执行程序，在屏幕上显示提示字符串或“?”号，请求用户响应从键盘上输入一行或多行字符串，并按顺序依次赋给各个字符串变量。

说明：

(1) LINE INPUT 语句中只能使用字符串变量，且至少要有一个，亦可以有多个串变量，各变量之间以逗号分隔，但在语句末尾不允许以逗号结束。

(2) 当有可选项 prompt<提示字符串>时，只在第一行输入时显示提示字符串，其它输入行只显示“?”号。

(3) 从键盘上输入的字符串不用双引号作界符，且行数必须与 LINE INPUT 语句中串变量的个数相等，一行字符串赋给一个变量。字符串行由任何可显示字符组成，并以回车键结束，其中包括双引号、空格及逗号等，双引号不再作为字符串界符，而是作为一个字符看待。

例 3.9

```
LINE INPUT prompt"a$,b$=";a$,b$
```

```
PRINT a$,b$
```

```
END
………………
OK. RUN
a$,b$ =How are you!
? STUDENT.
How are you !  STUDENT.
OK.
```

3.6.3 单键输入(GET KEY)语句

格式:GET KEY <数值变量>

功能:执行该语句时,暂停程序执行,等待用户从键盘上按下一个键,并将此键的 ASCII 十进制代码值赋给数值变量。

说明:在执行该语句时,光标消失,等待用户响应。当用户响应按某键后,按键字符亦不在屏幕上显示出来,而只是将此键代码值赋给数值变量。如果按下的键是 ASCII 码字符集中的字符,则 GET KEY 返回的是它的 ASCII 码;如果按下的键不是 ASCII 码字符集中的字符,而是键盘上的特殊键,则 GET KEY 返回的是特殊键的代码。

3.7 DATA/READ 语句与数据表恢复语句

在程序设计中,当需要数据量比较多时,如果采用 LET 语句,则需要许多 LET 语句,而使程序变得冗长。若使用 INPUT 语句,由于要输入的数据多,而使输入的时间长,且在输入过程中容易出错,如果输入过程中有一个数据出错,则必须重新输入,因此既费力又费时。为了方便有效地输入批量数据,True BASIC 提供了 DATA/READ 语句。

3.7.1 DATA/READ 语句

DATA/READ 两个语句配合使用。DATA 语句为 READ 语句的变量读入提供数据,READ 语句从 DATA 语句构成的数据表中读取数据。

(1) DATA 语句

格式:DATA <常量>[{<,常量>}]

功能:程序执行前,系统把 DATA 语句中的数据,按顺序在内存数据区中形成一个数据表,并设置一个数据指针,数据指针指向数据表中的第一个数据。

说明:

① DATA 语句在程序中的作用仅仅为 READ 语句提供数据,它本身并不做任何操作,是一个非执行语句,程序执行时会自动跳过 DATA 语句,不影响程序逻辑,所以它可以放在程序中的任何地方,但必须在 END 语句之前。

② DATA 语句中的常量可以是一个或多个,多个常量之间以逗号分隔。常量必须是不带运算符的原形数据,不得为变量、函数或表达式。需特别注意的是分数和带乘方的数,如 1/3 或 10^2 都是表达式,而负数则为合法的常量。

③ DATA 语句中如有字符串常量,如果字符串常量中首尾部带有空格,或其中含有逗号时,则该字符串常量必须用双引号括起来,否则可以省略双引号对。

④ 在程序中,可以将 DATA 语句中的多个常量写在一个 DATA 语句中,也可以将常量按原来顺序分散写在多个 DATA 语句中。

(2) READ 语句

格式:READ <变量>[{<,变量>}]

功能：当执行 READ 语句时，从内存的数据表中读取数据，并依次赋给 READ 语句中的各个变量。

在程序执行前，DATA 语句中的数据已经按顺序在内存数据区中形成了一个数据表，数据指针指向数据表中的第一个数据，程序执行时，READ 语句每读完一个数据，数据指针自动下移，指向下一个数据。指针指向的数据称当前数据，READ 语句总是读取当前数据，赋予相应变量，直到指针指向数据表末尾。

READ 语句中的变量可以是一个或多个，如有多个变量，则各变量之间以逗号分隔。多个变量可以写在一个 READ 语句中，也可以按原来顺序分散写在多个 READ 语句中。

在程序中使用 DATA 语句和 READ 语句时，要注意以下几点：

① 程序的所有 DATA 语句中常量的总数不得少于 READ 语句中变量的总数，否则会出现数据指针已经指向数据表末尾，使得 READ 语句中的某些变量因读不到数据而出错。

② 数据表中当前数据类型应与 READ 语句中对应变量的数据类型一致，规则与 INPUT 语句类同。

例 3.10 有 5 个学生，一门课的成绩分别为：87，89，70，76，80，编程求他们的总平均成绩。

```
DATA 87,89,70,76,80
READ a,b,c
READ d,e
LET aver=(a+b+c+d+e)/5
PRINT "average=";aver
END
..................
OK.RUN
average= 80.4
OK.
```

3.7.2 数据表恢复(RESTORE)语句

从以上介绍可见，数据表只能使用一次，READ 语句从数据表中读取数据，当数据指针已指向数据表末尾时，被读过的数据便不能再被读取和使用。如果需要重复使用这些被读过的数据，或其中一部分数据时，可使用 RESTORE 语句恢复数据表，而不必将 DATA 语句重写一遍。

数据表恢复语句的格式：

RESTORE

其作用是使数据指针重新指向数据表中的第一个数据，恢复数据表。

RESTORE 语句使数据指针重新指向程序中第一个 DATA 语句的第一个数据，从而使 RESTORE 语句之后的 READ 语句可以从第一个数据开始重新读取。

例 3.11

```
READ A,B                      数据表
DATA 1,2,3             指针——→1   A C H
RESTORE                       2   B D I
READ C,D,E,F,G                3     E J
RESTORE                       4     F
READ H,I,J                    5     G
DATA 4,5,6                    6
PRINT A,B,C,D,E,F,G,H,I,J
END
```

```
......................
OK. RUN
  1    2    1    2    3
  4    5    1    2    3
OK.
```

可见，借助 RESTORE 语句很容易实现重复使用数据表中的数据，而不必重写 DATA 语句。

RESTORE 语句可以恢复数据表，指针重新指向表中第一个数据，但有时只需要重复使用其中一部分数据，且并不一定从第一个数据开始，这时可以使用“重读”或“假读”的方法，使数据指针跳过不需要的数据而指向所需要重复使用的数据。

所谓“假读”是用程序中无用变量读取无用数据，移动指针。“重读”是用程序处理要用的变量，多次读取数据移动指针，最后一次读取才是真正有用的数据，它利用变量破坏性读入的特性，每读入一个数据代替原存的数据。

例 3.12

```
DATA 1,2,3,4,5                           数据表
READ a,b,c,d,e               指针——→    1  a f z
RESTORE                                  2  b f x
READ f,f,f,g,h                           3  c f z
RESTORE                                  4  d g y
READ z,x,z,y                             5  e h
PRINT a,b,c,d,e,f,g,h,x,y
END
....................
OK. RUN
  1      2      3      4      5
  3      4      5      2      4
OK.
```

程序中 READ 语句的变量 f 是用作“重读”，z 是用作“假读”的变量。

3.7.3 给变量提供数据的三种语句的比较

前面介绍的三种给变量提供数据的语句：LET 语句，INPUT 语句及 DATA/READ 语句，它们的共同点是均可使变量取得数据，但它们又有各自的特点。

LET 语句的主要优点是运算赋值，是使变量从表达式中取得数据的唯一方法。但一个 LET 语句只能将一个表达式计算赋值给一个变量或多个变量，当需要处理的数据较多时，则必须使用多个 LET 语句使得程序变得冗长，且原始数据变动时，需修改程序，程序通用性和灵活性差。因此，LET 语句适用于需要数据量较少的场合，在程序中往往用于给变量赋初值及运算赋值。

DATA/READ 语句可以把批量数据读入变量，但 DATA 语句中只能是常量，无运算功能，原始数据变动时，亦需修改程序。它适用于输入数据较多，而且这些数据不需经常变动的场合。

INPUT 语句可以给多个变量赋值，由于数据是在程序运行过程中从键盘上输入，因此当数据变动时，无需修改程序，只要重新运行程序，输入新的数据，程序虽然相同，但数据不

同，可获得不同的结果，所以对于类似计算公式或解决同类问题，程序具有通用性。但程序运行后需等待键盘输入数据，且对于任何一个输入错误，必须重新输入，故占用机器时间长，效率低，且 INPUT 语句亦无计算功能，从键盘上输入的只能是常量。它适用于输入的数据个数适量且需经常变动的场合。

由上可见，给变量提供数据的三种语句各有利弊，因此，在编写程序时应根据具体情况选用。

3.8 输出位置与数据格式控制

数据的输出格式是面对用户的数据显示或打印形式，为了适应不同的要求，选择适当的输出格式可以使输出的数据美观、清晰。

前面介绍的 PRINT 语句，虽然可以通过选用不同的列表位置控制符（,/;）来控制数据的输出位置，产生多种不同的数据输出格式，但由于列表位置控制符控制的输出位置基本是固定的，输出格式不够灵活，因此，在实际应用中，如果仅使用这些固有格式有时不能满足需要。本节将介绍如何改变显示区域宽度和屏幕的行宽，格式化输出的光标定位控制及如何按照要求的特定格式输出数据。

3.8.1 显示区域宽度和屏幕行宽的设置

True BASIC 启动时，系统自动将显示器的屏幕设置为 25 行，每行宽为 80 个字符，同时每行又等分为 5 个标准显示区域，每个标准域的宽度为 16 个字符。如果需要，程序可以改变显示区域的宽度和屏幕的行宽。

(1) 显示域宽设置语句

格式：SET ZONEWIDTH ＜数值表达式＞

功能：把显示区域的宽度设置为数值表达式的值。

(2) 屏幕行宽设置语句

格式：SET MARGIN ＜数值表达式＞

功能：把屏幕行宽设置为数值表达式的值。

(3) 说明

① 语句中数值表达式的值应为整数，如果为小数，则自动进行四舍五入取整。

② 行宽设置语句中数值表达式的值应在 1 到 80 之间。而且设置的屏幕行宽不能小于当前显示区域的宽度。

③ 域宽设置语句中数值表达式的值应在 1 到行宽之间，且不得超出行宽。

④ 带通道号的设置语句可用于打印机和文件行宽、域宽的设置。

(4) 查询显示域宽度和屏幕行宽语句

在设计数据的输出格式时，有时需要知道当前的显示域宽度和屏幕行宽，这时可以使用下列语句来查询：

查询域宽语句 ASK ZONEWIDTH ＜数值变量＞

查询行宽语句：ASK MARGIN ＜数值变量＞

上述两个语句的作用是分别将当前显示域宽度和屏幕行宽的值赋给相应的数值变量，返回程序。

例 3. 13

```
ASK MARGIN M
ASK ZONEWIDTH Z
RPINT "MARGIN=";M,"ZONEWITH=";Z
INPUT "Margin,Zonewidth=";x,y
SET MARGIN x
SET ZONEWIDTH y
ASK MARGIN M
ASK ZONEWIDTH Z
PRINT "MARGIN=";M,"ZONEWIDTH=";Z
END
…………………
OK. RUN
MARGIN=80     ZONEWIDTH=16
Margin,Zonewidth=60,10
MARGIN=60     ZONEWIDTH=10
OK.
```

执行完上述程序后，把屏幕行宽设置为 60，显示域宽度设置为 10。每行有六个显示域。

3. 8. 2 TAB 函数

在 PRINT 语句中使用列表位置控制符(,/;)控制数据的输出位置，在有的场合会很不方便，甚至相当困难，如打印函数曲线或过渡过程图时，输出位置是不断变化的，仅仅使用列表位置控制符控制输出不能满足要求。因此，True BASIC 语言提供了输出位置控制函数——TAB 函数，它可方便灵活地控制输出位置。

TAB 函数只能在 PRINT 语句中使用。在 PRINT 语句中使用 TAB 函数，可以使其后的各输出项在指定位置上输出。

格式：PRINT TAB(<数值表达式>);[<输出项表>]

功能：TAB 函数后的各数据按数值表达式的值所指定的位置上开始输出。

例 3. 14

```
PRINT TAB(5);3
PRINT TAB(10);5
END
………………
OK. RUN
    3
              5
    |         |
  第5列    第10列
```

说明：

(1) TAB 函数中的数值表达式的值为 1 到屏幕行宽间的整数，如为小数时，按四舍五入自动取整，代表输出位置(列数)。当输出位置超过屏幕行宽时，输出项将移到下一行在 TAB 函数所指定的位置上输出，此时输出位置为：数值表达式－屏幕行宽。

(2) TAB 函数与输出项之间以分号分隔，一个 PRINT 语句允许用多个 TAB 函数，TAB 函数间亦用分号分隔。但后一 TAB 函数中数值表达式的值应大于前一 TAB 函数中数值表达式的值，否则，移到下一行输出。如果两个 TAB 函数之间以逗号分隔，则在第一个输出项输出后，光标自动移到下一个域，如果第二个 TAB 函数所指定位置不小于当前光标所在的位置，则在所指定的位置输出第二个输出项，否则，移到下一行输出。例如：

```
OK. PRINT TAB(5);6,TAB(18);7
    6    7
OK. PRINT TAB(5);6,TAB(10);7
    6
         7
OK.
```

(3) TAB 函数与输出项之间应以分号分隔，则输出项按 TAB 函数中数值表达式所指定的位置输出；如果以逗号分隔，则输出项在指定位置的下一域输出，而 TAB 函数指定的位置

不起作用。例如：

```
OK. PRINT TAB(5),6
                        6
OK.
```

（4）一个 TAB 函数指定一个输出项的位置，如果 TAB 函数后有多个输出项，各输出项之间可以用逗号或分号分隔，其作用与 PRINT 语句中的逗号或分号相同。例如：

```
OK. PRINT TAB(5);6,-7,8;9
      6          -7          8  9
OK.
```

（5）PRINT 语句中 TAB 函数后可以不带输出项，如 PRINT TAB(X);，则 TAB 函数中的 X 指定程序中后继输出项的输出位置。例如：

```
PRINT TAB(5);
PRINT -7
END
………………
```

```
OK. RUN
        -7
OK.
```

例 3.15　某学生三门课的成绩分别为：语文 95，英语 80，计算机 90，编程输出该学生的成绩报告单。程序如下：

```
! Report of student Exam scores
!
PRINT TAB(15);"SCORES report"
PRINT
READ NAME $ ,NO $
PRINT TAB(5);"NAME=";NAME $ ;
  TAB(25);"NO. =";NO $
SET ZONEWIDTH 10
PRINT "CHINESE","ENGLISH",
PRINT "COMPUTER","AVERAGE"
READ CH,EN,CO
PRINT CH,EN,CO,
LET A=(CH+EN+CO)/3
PRINT A
DATA LI MING,25,95,80,90
END
………………
OK. RUN
              SCORES report

       NAME=LI MING         NO. =25
CHINESE ENGLISH COMPUTER AVERAGE
95        80        90        88.33
OK.
```

3.8.3　数据格式输出(PRINT USING)语句

PRINT 语句按行结构输出数据，无论一行只输出一个数据，或一行输出多个数据，都可用列表位置控制符(,/;)或用位置控制函数(TAB)控制数据的输出位置。另外，配合有 SET 语句设置输出行的行宽和域宽，基本满足了数据列表的要求，输出比较美观的图表。但美中不足的是，PRINT 语句对输出数据本身的格式无法控制，例如表格的标题或栏目，一般希望处于表格或栏目宽度的中间，或要求以左侧为准对齐，或相反以右侧为准对齐时，只用 PRINT 语句的位置控制，会相当繁琐；又如数值输出时，往往要求小数点对齐，有效位统一，或为防止输出表格被不合法篡改，希望在数值前不留空格并加上数符，钱数前加美元符号 $，或为使报表数据易读、易于核对、纠错，在数值中加入数位分隔符“,”号，或希望明确表示是某个物理量的值，物理量、数值连同单位一起输出……等等，只有位置控制功能，实现这类要求同样是相当困难。为此，True BASIC 语言提供了数据格式控制输出语句，它可以方便地实现各种要求格式的数据输出。

(1) 语句格式

PRINT USING <格式串表达式>:<表达式表>

PRINT USING <格式串表达式>:<表达式表>;

语句命令按<格式串表达式>所规定的格式,输出<表达式表>的值,可以是单个表达式的值,或用逗号分隔的多个表达式的值。第一个语句,当格式数据输出后,回车换行;第二个语句,光标位于最后一个数据的末尾,不回车换行,等待后继输出。PRINT USING 语句中的逗号,用作多输出项分隔符,不具有 PRINT 语句中按域位置控制作用,且在一个 PRINT USING 语句中,无论输出项有多少,只可能有一个分号,在语句末尾,表示后继数据在紧邻输出。注意,指定格式用的<格式串表达式>和代表输出数据的<表达式表>间用冒号分隔。如果只有一个格式串表达式,则语句中各输出项共享同一格式规定。

(2) 格式字符串

<格式串表达式>是由系统规定的格式符和其它字符组成的字符串,可以是字符串,或是已赋予格式串的字符串变量,亦可以是字符串表达式。系统规定的格式符有 11 个:〈、〉、+、一、$、%、#、*、,、. 和 ^ 。<格式串表达式>是 PRINT USING 语句的第一个输出项,因此其中可用其它字符,这些字符原样输出,同时,在<格式串表达式>中包含着若干个数据输出格式子串时,这些字符又起着格式子串的分隔作用。有关格式子串和输出数据的对应关系,在下面说明。

按输出数据不同,格式串又分为数值输出格式串和字符串输出格式串两类。

① 数值输出格式串。数值输出格式串用来指定数值输出格式,它所用的格式符有:首部符(+、一、$),给输出数值加数符或美元符“$”的前缀;数位符(*、%、#),一个数位符代表输出数值的一位数字和该数字在数中的位置;插入符(.、,、^),代表输出数值的小数点“.”的位置,或要插入数位分隔符“,”的位置和用科学表示法时指数部分的输出格式。

首部符:

+——负数前冠以“-”号输出,正数前冠以“+”号输出。

-——负数前冠以“-”号输出,正数前冠以空格符输出。

$——在数值前冠以“$”符输出。

数位符:任何一个数位符,代表一位数字,格式串中最少要有一个数位符,在多个数位符连用时,格式串代表输出数值的有效位。因此,可能出现数位符多于数值位数的情况,此时,不同的数位符,处理的方法不同。

对于整数部分,在不足位上:

——前置“”号

%——前置“0”

#——前置空格

对于小数部分,多余的数位符一律后置“0”。

注意,一个格式串中不允许使用不同的数位符,例如:“##%%”或“**#%”等都是不合法格式串。

表 3.4 中序号 4~8 是数位符少于数值位数的情况。当整数部分位数超过数位符(不包括单个首部符)时,输出和格式串等长度的一串“*”号,表示数值无法用该格式串指定的格式输出;当小数部分超出时,系统按指定有效位四舍五入截取输出。

表 3.4

序　号	数　值	格式串	输出结果
1	125.5	“＋###.##”	“＋125.50”
2	125.5	“－###.##”	“ 125.50”
3	125.5	“$＊＊＊.＊＊”	“$125.50”
4	25.543	“$###.##”	“$ 25.54”
5	25.543	“－%%%,%%”	“ 025.54”
6	25.543	“－＊＊＊。＊＊”	“ ＊25.54”
7	1125.5	“－###。##”	“＊＊＊＊＊＊＊”
8	25.543	“$$$#。##”	“ $25.54"

序号 4 中数值为 25.543，指定采用“$###.##”格式输出，因整数部分位数不足 3 位，多余的一个数位置成空格，输出结果为“$ 25.54”，如用%或＊作数位符，则输出结果为“$025.54”和“$ ＊25.54”。可见，在数值位数不定时，格式串由单个首部符和数位符构成很难有十分满意的结果，此时可用多个首部符来改善。在使用多个首部符时，输出时仍只有一个首部符前缀，但多余的首部符可代替数位符用，而且首部符代替的前缀总紧邻输出的数值，再有多余者前置空格，见表中序号 8。

切记，在数值格式串中整数部分，至少要有一个数位符。

插入符：格式串用“.”安排数值输出的小数点位置，如格式串无小数点，则表示输出整数。在数值输出时，格式串安排有“,”位置，输出时插入“,”号，作数位分隔用。输出在“,”相对应位的左右位上皆有数字，则插入“,”号，否则，“.”以空格替代输出。注意：“,”号不能插入多个首部符中间，否则显示“Badly formed USING String,”错误信息。

表 3.5

序　号	数　值	格式串	输出结果
1	32	“##,###”	“ 32”
2	1432	“##,###”	“ 1,432”
3	1e6	“##,###,###.##”	“ 1,000,000.00”
4	1432	“$#,###,###.##”	“$ 1,432.00”
5	1432	“$$,$##,###.##”	错误信息

表 3.6

序　号	数　值	格式串	输出结果
1	1.2e3	“###^^^”	“120e+1”
2	1.2e3	“#.##^^^”	“1.20e+3”
3	1.2e3	“#.##^^^^”	“1.20e+03”
4	1.2e3	“##.####^^^”	“12.0000e+2”
5	1.2e3	“+#.####^^^”	“+1.2000e+3”
6	1.2e3	“−−#.####^^^”	“ 1.2000e+3”

格式串中使用“^”号，可以用科学表示法输出数值，在使用“^”符时，至少要有 3 个“^”符，给 e+或 e−及 1 位指数留出位置，否则出错，多余的“^”符，在指数中前置“0”。指数部分必须在格式串的末尾，此时，全部数位符（包括小数点）表示科学表示法的数字部分，在数字部分位数不足时，小数点右移，减少指数值自动补足输出（见表 3.6 序号 1,4）；超出

时，小数点左移，增加指数值自动补足输出。因此，在有指数的格式串中，只要有一个首部符即可，即便使用多个首部符，亦不起数位符作用，只有多余的空格而已（见表 3.6 序号 6）。

格式串位数不足，数值无法输出的情况见表 3.7。

表 3.7

序　号	数　值	格式串	输出结果
1	1000	“###”	“***”
2	1000	“$$##”	“****”
3	1e1000	“#^^^”	“****”

② 字符串输出格式串。字符串数据由一定长度的字符组成，字符串数据输出格式符引用全部数值格式串的格式符。无论是数位符，首部符和插入符，每个符号代表一个输出字符。若格式串长度小于字符串长度时，全部输出“*”号；相反，如果格式串长度大于字符串长度，字符串取中输出，两边的多余格式符由空格填充；若多余格式符个数为奇数，则右侧空格比左侧多一个；此时，格式串可用字符串输出首部符“<” 和“>”号来调整，用“<”号表示字符串左调整输出，“>”表示字符串右调整输出。字符串输出首部符本身亦计入格式串长度。

表 3.8

字符串	格式串	输出结果
WXYZ	“########”	“ WXYZ ”
WXYZ	“*******”	“ WXYZ ”
WXYZ	“<%%%%%%%”	“WXYZ ”
WXYZ	“>$######”	“ WXYZ”
abcdefgh	“<-##.#”	“******”
abcdefgh	“>-###.###”	“ abcdefgh”
abcdefgh	“>+-$#%*.^,”	“ abcdefgh”

③ 格式串中其它字符：在格式串中可以加入除格式符外的其它字符，它们原样输出。语句中一个格式串对应多个输出项时，则各输出数据共享同一格式串。例如：

```
OK. PRINT USING “LENGTH=##.##CM”;25.5,4
LENGTH=25.50CM LENGTH= 4.00CM
OK.
```

在格式串中的其它字符还用作格式串子格式串的分隔符，一个子格式串对应一个输出数据。如果格式子串数比输出项多，则多余格式子串作废，但其中的非格式符字符还是要原样输出的；如果格式子串数少于输出项，则对应数据输出后，多余的数据项再重复使用从第一个格式子串开始的格式子串，依次按规定的格式输出数据。例如：

```
OK. PRINT USING "L1=--#.#CM L2=--#.##CM ";25.4,-4,5.5
L1=  25.4CM L2=  -4.00CM L1=     5.5CM L2=            CM
OK.
```

(3) 打印机输出

格式数据在打印机上输出时，首先应打开打印机通道，打开打印机通道的语句为：

```
OPEN #<通道号>;PRINTER
```

其中通道号为 1～999 之间的任意整数。再用下列格式的输出语句，就可在打印机上打印输出：

PRINT #<通道号>,USING <格式串表达式>:<表达式表>

打印完毕，使用语句 CLOSE #<通道号>关闭打印机。

3.8.4 USING $ 函数格式及使用

USING $ 函数的格式为：

USING $ (<格式串>,<表达式>[{<,表达式>}])

它的作用是将表达式的值(数值或字符串)，按格式串转换成格式化字符串返回。与所有函数一样，它可作为一个分量和其它字符串构成字符串表达式，或赋给字符串变量存储，供多次使用。

例 3.16

```
LET a $ ="#####.##"                    ..................
LET b=117.5                            OK. RUN
LET c $ =using $ (a $ ,b)                  117.50
PRINT c $                              OK.
END
```

3.8.5 全屏幕格式化输入和输出

以上介绍的 INPUT 语句、人机对话的提示、用户响应输入的字符和 PRINT 语句输出的数据都限制在背景窗中，严格地按行顺序显示，虽然 PRINT 语句提供了多种输出位置控制功能，但亦只能按行编辑显示数据。在许多实际应用中，例如菜单安排和选择，表格设计填写等，要求在全屏幕上按用户要求的格式，自由地编辑显示数据和文本，为此，True BASIC 提供了全屏幕格式化输入输出功能和语句。

(1) 全屏幕光标位置设置语句

格式：SET CURSOR <行,列>

它的作用是将光标位置设置在行、列所指定的坐标点上。其中行与列均为数值表达式，数值表达式的值自动取整，其取值范围分别是 1～25 行和 1～80 列。

光标是屏幕显示输入/输出字符的位置标志，按格式要求控制光标的屏幕位置，配合 INPUT，PRINT 语句就可实现格式化输入/输出。如果当前尚末进入全屏幕文本模式，执行光标全屏幕设置语句，将自动清屏，进入全屏幕文本模式，并将光标移到指定行、列位置，等待输入或输出显示。

说明：

① True BASIC 1.0 版本中，程序运行结束时，自动清屏，返回系统启动后的双窗口(编辑，背景窗)文本模式，为看清程序运行全屏幕显示的结果，必须使用暂停语句。语句格式为：

PAUSE <数值表达式>

其中，<数值表达式>的值是程序暂停的秒数。

2.03 版本无需使用 PAUSE 语句，程序运行结束，只有再按任意键后才返回双窗口文本模式。

② 建议不要在 25 行设置光标，否则在输入、输出后会引起屏幕上卷。

(2) 光标位置查询语句

格式：ASK CURSOR <行变量,列变量>

它的作用是将光标所在位置的行与列的当前值分别赋给行和列数值变量。

此外，使用语句 ASK MAX CURSOR<行变量，列变量>，则可查询显示窗口所允许的行列的最大值，其中行变量和列变量均为数值变量。

例 3.17

```
SET CURSOR 1,4
INPUT prompt "x,y=";x,y
SET CURSOR 2,5
PRINT x,y
PAUSE 4
END
..................
OK. RUN
```

运行上述程序时，自动清屏，在屏幕的第 1 行第 4 列显示“x,y=”信息，等待用户输入数据，然后在第 2 行第 5 列显示结果。

(3) 光标打开与关闭

在 True BASIC 启动后，光标处于打开状态，即在屏幕上有一个“_”号，标志光标所在的位置，如果需要，可以关闭光标，或重新打开光标。光标的打开与关闭可以使用光标开闭语句。其格式如下：

SET CURSOR <字符串表达式>

当字符串表达式的值为“ON”时，就打开光标；当字符串表达式的值为“OFF”时，则关闭光标。

此外，使用语句 ASK CURSOR<字符串变量>，则可以根据字符串变量的值来了解当前光标所处的状态(打开或关闭)。

小　结

(1) True BASIC 有两种基本类型数据：数值和字符串。每一类型的数据又有常量、变量、函数和表达式的形式。

(2) 数有两种表示法：一般表示法和科学表示法。一般表示法表示的数只能有 10 位有效位；科学表示法用以表示非常大或非常小的数。

字符串是包括在双引号中的字符序列，字符串长度最多可达 32767 个字符，双引号是字符串的界符，亦可以是字符串中的一个字符，但此时必须重复一次双引号对。字符串中英文字母大小写有别。

(3) 变量有数值变量和字符串变量两类，变量名命名规则为：以英文字母开头，其后可以是英文字母，数字或下划线“_”，变量名最多不得超过 31 个字符。字符串变量必须有“$”作后缀，以区别于数值变量。变量名中英文字母大小写等价。程序中引用变量是存或取变量中的数据，变量有破坏性写入、不破坏读出的性质。

(4) 表达式是用符合规定的运算符及括号将常量、变量及函数连接起来的处理式，数值运算的优先级为：括号→函数/乘方→乘/除→加/减。字符串处理的优先级为：括号→函数→截取→连接。

(5) True BASIC 提供了一些标准函数和子程序供用户调用，函数可以在表达式中直接调用，只需给出函数名及自变量的值即可返回调用结果，而子程序则必须使用 CALL 语句调用。

(6) 一个 True BASIC 程序由一系列简单语句和控制结构组成，且必须以 END 语句结束。到目前为止，已介绍的简单语句有：

① 三种给变量提供数据的语句（LET，INPUT 和 DATA/READ）；

② 数据输出语句（PRINT），数据格式输出语句（PRINT USING），行宽、域宽设置和查询语句，光标位置设置和查询语句；

③ 恢复数据表语句（RESTORE）；

④ 注释语句（REM）；

⑤ 结束语句（END）；

⑥ 数据格式输出语句（PRINT USING）。

习　题

1. 用科学表示法表示下列数

(1) 7.35×10−9

(2) −127879607457

(3) 10^{-81}

(4) 10000000

(5) 0.0008291

(6) −0.000000128

2. 指出下列变量是否合法，如不合法，请指出不合法原因：

(1) 8NAME $

(2) AGE9

(3) ELSE

(4) NUMBER-ONE $

(5) 1E+09

(6) “ABC”

(7) MAXNUM

(8) Z $ 3

(9) LARGE-ONE $

(10) PI

3. 下列代数式用 True BASIC 表达式表示：

(1) $\frac{88-52\times 63}{18\times 47\div 3}$

(2) $\frac{(2\sin(45^\circ))^4}{e^2\ln 5}$

(3) $\frac{1}{\frac{1}{a}+\frac{1}{b}}$

(4) $G\frac{m_1m_2}{r^2}$

(5) $(X+Y)(X-5)$

(6) $1+\sin^2X$

(7) $\mathrm{tg}X+3\sqrt{X}$

(8) $(U+V)^{i-1}$

(9) $\frac{a\sin 64^\circ}{2b}$

(10) $(XY)^2+\frac{5}{X}$

4. 从字符串“STUDENT”中截取下列子字符串：

(1) 字符串“E”；

(2) 字符串“DEN”；

(3) 字符串“SUN”。

5. 已知 A $ = “PEOPLE'S”，B $ = “REPUBLIC”，C $ = “CHINA”

(1) 用字符串连接操作获得字符串“PEOPLE' S REPUBLIC OF CHINA”。

(2) 用字符串截取子字符串和字符串连接操作，从上述三个字符串中获得字符串“CHINESE”。

6. 写出下列函数的结果

(1) ANGLE(1,1)

(2) ABS(−1.45)

(3) LOG10(100)

(4) LOG2(1024)

(5) SGN(4,6)

(6) SGN(−4,6)

(7) MOD(−9,2)

(8) MOD(9,−2)

(9) INT(8.7)

(10) INT(−4.3)

(11) ROUND(149.28,1)

(12) ROUND(149.28)

(13) ROUND(149.28,−1)

(14) ROUND(149.28,-2)

(15) TRUNCATE(149.28,1)

(16) TRUNCATE(149.28,-2)

7. 指出下列语句的错误并改正：

(1) X=5,Y=3

(2) LET X=5:Y=3:Z=7

(3) LET X+2=Y+5

(4) LET A$=FLOG

(5) LET B="ABC"

(6) LET X=Y=Z=A+B

(7) PRINT X=A+B

(8) INPUT "A,B,C=";A,B,C

(9) DATA A,B$,5%

(10) DATA -3,1/3,2^3

8. 写出下列程序的运行结果

(1)

```
READ A,B
DATA 1,2,3
READ C,D,E
RESTORE
DATA 4,5,6,7
READ F,G,H,I,J,K,L
RESTORE
READ M,N
DATA 8,9,10
PRINT A,B,C,D,E,F,G,H,I,J,K,L,M,N
END
```

(2)

```
READ A,B,C,D,E,N
LET T=A+B+C+D+E
LET V=A/N
PRINT "MEAN=";V
DATA 8,4,6,15,4,5
END
```

9. 已知三角形的三边分别为 a=16,b=17,c=19.5,求三角形的面积。三角形面积的公式为：$S=\sqrt{p(p-a)(p-b)(p-c)}$,其中 $p=(a+b+c)/2$。

10. 编写一个程序,从键盘上输入一个正数,求以此数为半径的球的体积 V 和表面积 S,

$$V=4\pi R^3/3, S=4\pi R^2$$

11. 编写一程序,求某学生的总分和平均分(门数自定),并打印出该学生的成绩报告单。

12. 请写出一个把华氏温度转化为摄氏温度的程序。公式为：C=5/9(F-32),其中 F 是华氏温度值,C 是摄氏温度值。求当 F=32,68,59,52,80 时的 C 值。

13. 编写程序打印如下图形

```
      *
    * * *
  * * * * *
* * * * * * *
```

14. 已知数据 A=3.457, B=0.145,C=14.189,D=42.1,要求分行输出各数据,且小数点对齐,小数点后保留两位小数。试编此程序。

15. 已知梯形上底、下底和高分别为 L1=5cm, L2=10cm,H=4cm,编程求此梯形的面积,并输出带单位的上底、下底、高及面积的值。

16. 写出下列程序的运行结果：

(1)

```
READ a,b,c,d,e,f,g,h
PRINT USING "$$#####.###";
    a,b,c,d,e,f,g,h
DATA 0.936,0.9999,1.5792,12.3456
DATA 234.5763,1234,12345,1234567
END
```

(2)

```
PRINT USING "##.##^^^^";2345.67
 PRINT USING "#.###^^^^";18888
PRINT USING "+##.##^^^";12345
 PRINT USING "#,###.##^^^";1234.5
END
```

(3)

```
READ A$,B$
PRINT USING "######";A$,B$
PRINT USING "<###.##";A$
PRINT USING ">###.##";B$
DATA abcd,efg
END
```

第 4 章　判断选择结构与循环结构

True BASIC 提供了另外两种程序控制结构。一种是判断选择结构，根据判断给定的条件是否满足而控制程序的走向，选择执行某些语句或另外一些语句，完成不同功能；另一种是循环结构，循环是控制程序转回执行，重复执行某个语句块完成某项功能。而每次执行时，语句虽类同，但处理数据不同而获得不同结果。循环结构是根据给定的条件决定继续执行循环还是退出循环。

判断选择结构赋予程序逻辑判断功能，循环结构则是充分发挥计算机低能高速的特性，以简单的重复完成复杂的处理，它也必须包含逻辑判断功能。逻辑判断用的条件是由关系表达式或逻辑表达式来描述的。

4.1　逻辑条件与逻辑表达式

4.1.1　逻辑命题，逻辑值

逻辑学是一门有关推理和论证的科学。在日常生活中，我们遇到的逻辑命题很多，例如：判断一件事情的真与假，判断某个论点的正确与错误，或判断某个答案的对与否，实际上这些都是逻辑命题。例如：A 小于 B 或小于 C，如果这个逻辑命题成立，称为“真”，否则称为“假”。这个逻辑命题在 True BASIC 中可用逻辑表达式表示如下：

A < B OR A < C

逻辑表达式的运算结果是一个逻辑值，而不是数值或字串。

逻辑值只有两个，即“真”和“假”。True BASIC 语言中没有逻辑型数据，但在机器内部，表示“真”与“假”一般是用数值代表。例如，“假”值一般用 0 表示，“真”值一般用 1 表示。

逻辑命题实际就是逻辑判断用的条件。在 True BASIC 中可用关系表达式来表示逻辑条件或用逻辑表达式来表示复杂的逻辑条件。

4.1.2　关系表达式

关系表达式是用关系运算符把两个表达式连接起来的式子，即

< 表达式 1 > < 关系运算符 > < 表达式 2 >

其中，表达式是数值表达式或字符串表达式。表达式 1 和表达式 2 必须类型一致。关系表达式是描述两个同类表达式的比较逻辑。

True BASIC 的关系运算符及它们的含义见表 4.1。例如：X$ < Y$ ，A > 200 ，A+B >= C ，NAME$ = "GAO" 和 2*X+3 < Y+5 均是合法的关系表达式。

关系表达式的运算结果为“真”或“假”两逻辑值。当表达式的值满足关系运算符所表示的关系时，其逻辑值为“真”，否则为“假”。

关系表达式按操作对象是数值或字符串分为数值关系表达式和字符串关系表达式。

表 4.1　关系运算符

关系运算符	>	<	=	>=	<=	<>
含　义	大于	小于	等于	大于等于	小于等于	不等于
注	单　键			复　合　键		

(1)数值关系表达式。是用关系运算符把两个数值表达式连接起来的式子，数值关系表达式举例见表 4.2。

表 4.2　数值关系表达式举例。

例	解　　释
16+31<510	结果为真
N<=50	如变量 N 的值不大于 50 为真，否则为假
A*B=270	表达式 A*B 的值等于 270 为真，否则为假
A+B>=C	表达式 A+B 的值不小于 C 为真，否则为假
B*B-4*A*C>0	表达式 B*B-4*A*C 的值大于 0 为真，否则为假
I<>J	变量 I 的值不等于变量 J 的值为真，否则为假

数值关系表达式的运算顺序是：先计算数值表达式的值，然后再进行比较，其运算结果是一个逻辑值。关系运算符两边的数值，若满足运算符的关系，则关系式取“真”值，否则取“假”值。例如，关系式 5*2<87 的运算结果为“真”值，而 18-6 > 24 的运算结果为“假”值。

由于实数在计算机中是近似表示的，所以在理论上应当相等的量，计算机却可能认为不等。因此对两个实数 a,b 进行比较时，最好不用“=”这个关系运算符，而把关系表达式改写为：

ABS (a-b) < 1E-5

即根据具体情况给出一个很小的值(如 0.00001)，只要两个量之差的绝对值小于此值，就认为它们相等。

(2)字符串关系表达式。是用关系运算符把两个字符串表达式连接起来的式子，称为字符串关系表达式。例如：A$ >= B$，D$ =“JANUARY”，E$[2:4]=“GOOD”和 F$ <> G$均为合法的字符串关系表达式。

字符串关系表达式的运算结果是一个逻辑值。运算顺序是：先进行字符串表达式的运算，再进行字符串比较运算。字符串关系表达式举例见表 4.3。

表 4.3　字符串关系表达式举例

字符串关系表达式	逻辑值	解　　释
“ABCD”<“BCDE”	T	字母 A 的 ASCII 码值小于字母 B 的 ASCII 码值
“JUNE”=“JUNE”	T	等号前后两个字符串的所有字符依次相等
“JUNE”<“JULY”	F	因 N 的 ASCII 码值大于 L 的 ASCII 码值
“EFGH”<“EFGHUV”	T	两串的前 4 个字符相等，但后者串长比前者长n
“YES”=“YES”	F	空格也作为字符参与比较，所以两串不等
“AB”&“CDEF”=“ABCDEF”	T	因“AB”&“CDEF”连接结果是“ABCDEF”
“CDEFGH”[3:5]=“EFG”	T	因“CDEFGH”[3:5]求得子串为“EFG”
“THAT　IS”=“THATIS”	F	虽两串中字母相同，但前者串中有空格

注：表中 T 表示逻辑值“真”，F 表示逻辑值“假”。

字符串比较是从两个串的左端开始，逐个字符按它们的 ASCII 码值比较，代码大的为

大，代码小的为小。两个字符串相等，必须所有字符依次对应相等，长度相等，两个空串必定相等；如果不相等，以最早出现的第一个不同的字符的比较结果而定；若两个字符串长度不同，而短串的所有字符又与长串前面相应的字符都相等，则长度长的为大；空格也作为一个字符参与比较。注意：大小写英文字母 ASCII 码值不相等，字符串中字母大小写有别。

注意：(1)不能在复合运算符中间放入空格，如：“>=”不能写成“> =”。(2)关系运算符所连接的变量的类型必须一致，即不能把一个数值型变量与一个字符串型变量进行比较。

4.1.3 逻辑表达式

True BASIC 中没有逻辑常数，没有逻辑变量，但有逻辑表达式。

逻辑表达式是关系表达式通过逻辑运算符连接起来的逻辑式。它的格式为：

< 关系表达式 L1 > < 逻辑运算符 > < 关系表达式 L2 >

其中，< 逻辑运算符 > 有 AND、OR、NOT 三种。例如：

A+B > C AND B+C > A AND C+A > B

(MOD(Y,4)=0 AND MOD(Y,100)<>0) OR MOD(Y,400)=0

这两个都是合法的逻辑表达式，其中 A,B,C,Y 为数值变量。

逻辑表达式的运算结果(即关系表达式通过逻辑运算的结果)仍然是一个逻辑值。

逻辑运算的规则如下：

假设 L1 与 L2 分别表示一个关系表达式，则逻辑表达式 L1 AND L2，只有在关系式 L1 与 L2 同时为“真”时，其值才为“真”，否则为“假”，称逻辑与。

逻辑表达式 L1 OR L2，只要 L1 或 L2 的值中有一个为“真”，其值为“真”；只有当 L1 与 L2 的值均为“假”时，其值才为“假”，称逻辑或。

逻辑表达式 NOT L1 表示对 L1 的逻辑值取反，即当 L1 的值为“真”时，NOT L1 的值为“假”，L1 的值为“假”时，NOT L1 的值为“真”，称逻辑非。

上述的逻辑运算规则综合如表 4.4 所示。

表 4.4 逻辑运算真值表

L1 的值	L2 的值	L1 AND L2	L1 OR L2	NOT L1	NOT L2
T	T	T	T	F	F
T	F	F	T	F	T
F	T	F	T	T	F
F	F	F	F	T	T

注：表中 T 表示逻辑值“真”，F 表示逻辑值“假”。

表 4.5 逻辑表达式举例

逻辑表达式	逻辑值	解　　释
8 * 4>12+9 OR 13+24=15	T	L1 的值为 T，L2 的值为 F，所以 L1 OR L2 的值为 T
13−8<15 AND 8+3<>21	T	L1 的值为 T，L2 的值为 T，故 L1 AND L2 的值为 T
NOT 16<19	F	L2 的值为 T，NOT L2 的值为 F
“AB”>“EF” OR NOT “b”<“B”	T	L1 的值为 F，L2 的值为 T，故 L1 OR L2 的值为 T
NOT (10>9) AND 17<18	F	L1 的值为 F，L2 的值为 T，故 L1 AND L2 的值为 F

在进行逻辑表达式的运算时，运算的顺序依次是算术运算，字符串运算，关系运算与逻辑运算，而逻辑运算的优先次序则是 ( )，NOT，AND，OR。

4.2 判断选择结构

判断选择结构是根据所给条件成立与否，即根据逻辑表达式的值是“真”或“假”决定程序的走向，有选择地执行某些语句。True BASIC 提供了判断选择语句和判断选择结构，可根据问题的需要选择使用。

4.2.1 单行 IF－THEN 语句

格式：(1) IF ＜条件＞ THEN ＜语句＞

(2) IF ＜条件＞ THEN ＜语句 1＞ ELSE ＜语句 2＞

功能：格式 1 当＜条件＞满足时，执行 THEN 后面语句，然后执行该语句行的下一行语句；当＜条件＞不满足时，则直接执行该语句行的下一行语句。

格式 2 当＜条件＞满足时，执行 THEN 后面的语句 1；＜条件＞不满足时，执行 ELSE 后面的语句 2，不论执行语句 1 或语句 2，在这之后均执行本语句的下一行语句。

单行 IF－THEN 语句由三部分组成：

① 语句定义符 IF－THEN 或 IF－THEN－ELSE。

② ＜条件＞是关系表达式或逻辑表达式，其值为“真”或“假”，“真”代表条件满足，“假”代表条件不满足。

③ THEN 或 ELSE 后面的语句必须是单行可执行语句，不能是 REM、DATA、DIM、END 等非执行语句，更不能是选择结构或循环结构。单行 IF－THEN 语句不允许嵌套。

例 4.1 由键盘输入一个数 X，当 X>0 时，计算 SQR(X)，否则结束。

```
! count sqr(x)                                oK. run
INPUT prompt “Enter a number:”;x              Enter a number:25
IF x>0 then PRINT sqr(x)                        5
END                                           ok.
......................
```

例 4.2 测算婴儿身长，取决于出生身长 L0(厘米)和月龄 M，未满一周岁婴儿身长测算公式为：

$$L=L0+2.5M \qquad (M<=6)$$

$$L=L0+2.2M \qquad (M>6)$$

```
! count length
INPUT prompt ″L0,m=″:L0,m
IF m<=6 then LET l=l0+2.5 * m else LET l=l0+2.2 * m
PRINT ″Month:″;m,″Length:″;l;″ (cm) ″
END
......................
ok. run
L0,m=56,5
Month:5            Length:68.5(cm)
ok.
```

例 4.3 试编写计算下列函数 Y 的程序：

$$Y=\begin{cases}\dfrac{\cos(X)}{1+X} & (\text{当 } X>=0\text{ 时})\\ e^{-x^2} & (\text{当 } X<0\text{ 时})\end{cases}$$

```
! Find solution of the fnction for given x
INPUT prompt "Enter the value of x:";x
IF x>=0 then LET y=cos(x)/(1+x) else LET y=exp(-x^2)
PRINT "x=";x,"y=";y
END
......................
ok.run
Enter the value of x:2
x=  2          y=-.138716
ok.run
Enter the value of x:-1
x=-1           y= .367879
ok.
```

4.2.2 多行 IF－THEN 结构

单行 IF－THEN－ELSE 语句，无论条件是否满足都只能执行单个语句，限制较多，为扩展功能引入 IF－THEN－ELSE 结构。

```
格式 1：  IF <条件> THEN
               ……语句块 1……
               END IF
格式 2：  IF <条件> THEN
               ……语句块 1……
          ELSE
               ……语句块 2……
               END IF
```

功能：当<条件>满足时，则执行 THEN 后面的语句块 1；当<条件>不满足时，忽略语句块 1，对格式 1，则直接执行 END IF 后面的语句。对格式 2，则执行 ELSE 后面的语句块 2。无论执行语句块 1 还是语句块 2，在这之后继续执行 END IF 后面的语句。

说明：(1) 多行 IF－THEN 结构中的语句块，可以是单行语句或多行语句块，语句块中可以包括判断选择结构和循环结构。(2) 多行 IF－THEN 结构中，最后一个语句必须是 END IF 语句。

例 4.4 三数排序。由键盘输入三个任意数，按从大到小的顺序输出。

解 先从输入的三个数中找出最大的放在最前面，并输出该数；再从剩下的两个数中找出较大者，并输出该数和最后一个数。

```
! sorting data
INPUT n1,n2,n3
IF n1>max(n2,n3) then
  PRINT n1,max(n2,n3),min(n2,n3)
ELSE
  PRINT max(n2,n3),
  IF n1>min(n2,n3) then
    PRINT n1, min (n2,n3)
  ELSE
    PRINT min(n2,n3),n1
  END IF
END IF
END
......................
ok. run
```

```
? 46,39,98
98          46          39
ok. run
? 589,453,765
765         589         453
ok.
```

例 4.5　从键盘上输入 a，若 a=1 或 2，打印 a 的值；否则打印 a，并计算、打印 1/(a-1)+1/(a-2)的值。

```
! Print the value of y
INPUT prompt"a=";a
IF a=1 or a=2 then
  PRINT "a=";a
ELSE
    LET y=1/(a-1)+1/(a-2)
    PRINT "a=";a,
    PRINT "1/(a-1)+1/(a-2)=";Y
END IF
END
......................
ok. run
a= 1
a= 1
ok. run
a= 4
a= 4    1/(a-1)+1/(a-2)= .833333
ok.
```

4.2.3　选择结构的嵌套

选择结构的嵌套是指在选择结构中又包含有选择结构或单行的 IF-THEN 语句。

单行 IF-THEN 语句和 IF-THEN-ELSE 语句中不能嵌套 IF-THEN 语句。在 IF-THEN 结构和 IF-THEN-ELSE 结构的语句块中，可以嵌套 IF-THEN 语句和 IF-THEN-ELSE 语句，它们可以嵌套在 THEN 后的语句块中，也可以嵌套在 ELSE 后的语句块中。

下面是选择结构嵌套的程序举例。

例 4.6　计算三角形面积。由键盘输入三条线段的长度，求这三条线段所构成三角形的面积。

解　要计算三条线段所构成三角形的面积，首先要判断这三条线段是否能构成三角形，若能构成三角形，则三条线段的长度必须满足：

(1)均大于 0；(2)两边之和大于第三边。

```
! calculate the area of the triangle
PRINT "The length of three sides for a triangle:";
INPUT a,b,c
IF a>0 and b>0 and c>0 then
  IF a+b>c and b+c>a and c+a>b then
    LET s=(a+b+c)/2
    LET s=sqr(s*(s-a)*(s-b)*(s-c))
    PRINT"a=";a,"b=";b,"c=";c,"s=";s
  ELSE
    PRINT "no result"
  END IF
ELSE
  PRINT "no result"
END IF
END
......................
ok. run
The length of three sides for a triangle :5,4,8
a= 5    b= 4    c= 8    s= 8.18153
ok. run
The length of three sides for a triangle :
 -4,-9,2
no result
ok.
```

在这个例子中，判断选择结构是嵌套在 THEN 后面的语句块中。例 4.4 中，IF－THEN－ELSE 语句嵌套在 ELSE 后的语句块中。

例 4.7　美国某大学的学费付费办法：本地的学生选课 12 学分及 12 学分以下付 350 元，12 学分以上每增加一个学分增付 20 元；外地的学生选课 12 学分及 12 学分以下付 800 元，12 学分以上每增加一个学分增付 45 元。输入某学生的编号、姓名、选课学分及该学生是否为本地学生等信息。编程计算该学生应付的学费。

解　学费计算，有四种情况：

（1）本地学生，选课学分＜＝12，付费 350 元。

（2）本地学生，选课学分＞12，付费 350＋（总学分－12）＊20 元

（3）外地学生，选课学分＜＝12，付费 800 元

（4）外地学生，选课学分＞12，付费 800＋（总学分－12）＊45 元

这四种情况由是否为本地学生及选课学分的多少两个条件来决定。因此采用嵌套的多行 IF 语句来实现较为方便。

程序中所用变量如下：SN 为学生的编号；NAME $ 为学生的姓名；C 为特征值（C＝1 表示为本地学生，C＝2 表示为外地学生）；U 为选课学分；T 为应付的学费。

```
! Compute tuition fee for student
INPUT prompt“Enter serial number   :”:sn
INPUT prompt“Enter name       :”:name $
INPUT prompt“Enter characteristic value:”:c
INPUT prompt“Enter number of credits   :”:u
IF c=1 or c=2 then
  IF c=1 then
    LET t1=350+20 * (u-12)
    IF u<=12 then LET t=350 ELSE LET t=t1
  ELSE
    LET tl=800+45 * (u-12)
    IF u<=12 then LET t=800 ELSE LET t=t1
  END IF
  PRINT“SERIAL number:”;sn
  PRINT“NAME              :”;name $
  PRINT“CREDITS           :”;u
  PRINT“TUITION fee= $ ”;t
ELSE
  PRINT“WRONG characteristic data”
END IF
END
..............................
ok. run
Enter serial number        : 0123
Enter name            :Zhang ling fu
Enter characteristic value   : 2
Enter.number of credits       : 11
SERIAL number : 123
NAME              ;Zhang ling fu
CREDITS              : 11
TUITION fee= $ 800
ok.
```

本例中有三层 IF 语句嵌套，单行的 IF－THEN 或 IF－THEN－ELSE 语句只能嵌套在判断选择结构中。

4.2.4　嵌套与逻辑表达式

当解决有多个条件判断选择的问题时，可用嵌套的选择结构来实现。嵌套层次的多少取决于条件的多少。多个条件往往可用一个逻辑表达式来表示，从而可减少或避免选择结构嵌套，简化程序，提高程序的可读性。

例 4.8　输入一个年份，判断该年是否为闰年，并输出相应的信息。

解：　判断某年是否为闰年的具体条件是：

（1）　若年号能被 4 整除，但不能被 100 整除的是闰年。

(2) 若年号能被100整除且能被400整除的是闰年。

解此题可有几种不同的编程方案，下面例举两种方案分析比较。

方案1：

```
! Find the leap year
INPUT prompt “Enter year:”;y
LET a4=mod(y,4)
LET a100=mod(y,100)
LET a400=mod(y,400)
IF a4=0 then
  IF a100=0 then
    IF a400=0 then
      LET c=1
    ELSE
      LET c=0
    END IF
  ELSE
    LET c=1
  END IF
ELSE
    LET c=0
END IF
IF c=1 then
    PRINT year;“is a leap year.”
ELSE
    PRINT year;“is not a leap year.”
END IF
END
……………………
OK. RUN
Enter year: 2000
2000 is a leap year.
ok. run
Enter year: 2003
2003 is not a leap year.
ok. run
Enter year:2100
2100 is not a leap year.
ok.
```

方案2：

```
! Find the leap year
INPUT prompt “Enter year:”;y
LET a4=mod(y,4)
LET a100=mod(y,100)
LET a400=mod(y,400)
IF a4=0 and a100<>0 or a400=0 then
    PRINT y;“is a leap year.”
ELSE
    PRINT y;“is not a leap year.”
END IF
END
```

方案1用了三层嵌套的选择结构，方案2将这三个条件写成一个逻辑表达式：

(MOD(Y,4)=0 AND MOD(Y,100)<>0) OR MOD(Y,400)=0

方案2和方案1的运行结果完全一样。虽然表示条件的逻辑表达式复杂了，但程序更为简洁，问题是要设计出能反映多个条件的逻辑表达式。因此在进行多个条件的选择结构程

序设计时，必须在嵌套层次的多少与逻辑表达式的复杂性方面进行综合考虑，选取一个适合的方案。

4.3 多分支判断选择结构

在实际应用中，往往要求多条件多分支，在满足不同条件时，执行不同语句块，完成不同的功能。IF－THEN 结构嵌套可以解决多分支判断选择问题，请看下面例子。

例 4.9 猜数游戏。利用随机函数 RND 产生 10～16 之间的随机整数。若猜对了，或者猜错了，或者猜的数超出了给定的范围(小于 10 或大于 16)，都要给出相应信息。

```
! progran to play a guessing game
RANDOMIZE
LET answer=int(rnd * 7)+10
PRINT “thinking of a number from 10 to 16.”
INPUT prompt “Enter your guess:”;guess
IF guess<10 then
  PRINT “must be at least 10.”
ELSE
  IF guess>16 then
    PRINT “Can't be more then 16.”
  ELSE
    IF guess<answer then
      PRINT "Too low."
    ELSE
      IF guess>answer then
        PRINT "Too high."
      ELSE
        PRINT "Correct !!"
      END IF
    END IF
  END IF
END IF
PRINT "The number was";answer;"."
END
......................
ok. run
Thinking of a number from 10 to 16
Enter your guess:14
Too high
The number was 12
ok. run
Thinking of a number from 10 to 16
Enter your guess:9
Must be at least 10
The number was 15
ok.
```

本例是一个多分支的情况，程序中用了一系列的 IF 语句，后一个 IF 语句都嵌套在前一个 IF 语句中。嵌套的 IF 语句虽能解决问题，但难读，而且相对于每个 IF 语句都需要有一个 END IF 语句配对。过多的 END IF 语句使程序复杂化，也容易丢失出错。

True BASIC 提供了一个更好方法，即多分支选择结构 IF－THEN－ELSE IF 语句(也称为 ELSE IF 语句)，更适合于解决此类问题。

4.3.1 IF－THEN－ELSE IF 语句

格式：

```
IF <条件 1> THEN
  ……语句块 1………
  ELSE IF <条件 2> THEN
  ……语句块 2………
          ⋮
  ELSE IF <条件 n> THEN
  ……语句块 n………
```

```
[ELSE
……语句块 n+1………]
END IF
```

功能：按从上到下的顺序逐个检查各个<条件>，如<条件>为真，则执行该<条件>相应的语句块，然后执行 END IF 后面的语句。如果所有的<条件>为假，则执行 ELSE 后面的语句块，再执行 END IF 后面的语句，若不含 ELSE，就直接执行 END IF 后面的语句。

说明：(1) ELSE IF 语句可以有任意多个。(2) ELSE 部分可以省略。

例 4.10　用 IF—THEN—ELSE IF 语句来重写例 4.9 中的猜数游戏。

```
! progran to play a guessing game
RANDOMIZE
LET answer=int(rnd * 7)+10
PRINT "thinking of a number from 10 to 16."
PRINT
INPUT prompt "Enter your guess:";guess
IF guess<10 then
  PRINT "must be at least 10."
ELSE if guess>16 then
  PRINT "Can't be more then 16."
ELSE IF guess<answer then
  PRINT "Too low."
ELSE IF guess>answer then
  PRINT "Too high."
ELSE
  PRINT "Correct !!"
END IF
PRINT "The number was";answer;"."
END
```

可见用 ELSE IF 结构替代 IF—THEN 结构嵌套编写的多分支程序要简洁，可读性强。

例 4.11　500 元以上为大面额存款，分 1 年、2 年、3 年、5 年、8 年五档储蓄年限，其月利息分别为 0.58%、0.67%、0.78%、0.96%、1.2%，不满一年的按活期利息 0.24%计算。没有档次的年限，按最近的档次累计计算（如 7 年可按 5 年+2 年档次计算）。试编写一个程序，到期应拿本金和利息若干？

解　采用键盘输入本金和储蓄年限，本金输入必须大于或等于 500 元，年限必须是 1～8 之间的整数。若不满 1 年的活期可任意输入 0～11 之间的整数表示月份。

```
! computer
PRINT "enter s(>=500),year (1-8):";
INPUR s,year
IF year=1 then
  LET s=s * (1+0.0058)^ 12
ELSE if year=2 then
  LET s=s * (1+0.0067)^ 24
ELSE IF year=3 then
  LET s=s * (1+0.0078)^ 36
ELSE if year=4 then
  LET s=s * (1+0.0078)^ 36
  LET s=s * (1+0.0058)^ 12
ELSE if year=5 then
  LET s=s * (1+0.0096)^ 60
ELSE if year=6 then
  LET s=s * (1+0.0096)^ 60
  LET s=s * (1+0.0058)^ 12
ELSE if year=7 then
  LET S=S * (1+0.0096)^ 60
  LET S=S * (1+0.0067)^ 24
ELSE IF year=8 then
  LET s=s * (1+0.0120)^ 96
ELSE
  INPUT prompt "enter month=:";month
  LET s=s * (1+0.0024)^ month
END IF
IF year>=1 and year<=8 then
  PRINT "YEAR=";YEAR
ELSE
  PRINT "MONTH=";MONTH
END IF
PRINT "S=";S
```

```
END
........................
ok.run
Enter s(>=500),year(1-8): 500,3
YEAR= 3
S= 661.373
ok.run
Enter s(>=500),year(1-8):1000,5
YEAR= 5
S= 1774.03
Enter s(>=500),year(1-8):10000,8
YEAR= 8
S= 31428.9
OK.
```

4.3.2 SELECT CASE 选择语句

这也是一种多分支选择结构。在对同一表达式判断时，SELECT CASE 比 ELSE—IF 更简洁。

```
格式： SELECT CASE <表达式>
           CASE <TEST11> , <TEST12> ,……
               ……语句块 1……
           CASE <TEST21> , <TEST22> ,……
               ……语句块 2……
                       ⋮
           [CASE ELSE
               ……语句块 E……]
           END SELECT
```

功能：计算表达式的值，将此值从上到下、从左到右的顺序与 CASE 检测条件相比较。如找到了相匹配的检测条件，就执行与该检测条件相应的语句块。然后跳到 END SELECT 的后继语句继续执行。如果没有相匹配的检测条件，则执行 CASE ELSE 后的语句块 E，接着去执行 END SELECT 的后继语句。

说明：

(1) CASE <TEST>,…… 语句模块可以有任意多个。CASE ELSE 可以省略，在 CASE ELSE 省略时若没有匹配的 <TEST>时，则直接跳出 SELECT CASE 结构，执行 END SELECT 的后继语句。

(2) <表达式>可以是数值表达式或字符串表达式。<TEST>为检测条件，检测条件均为常量，它必须与<表达式>的类型一致。

(3) <TEST>在 CASE 语句中可以有一个或多个，它们之间用逗号“，”隔开。

<TEST>检测条件可以是下述三种形式：

① <常量表>

<表达式>的值是常量表中元素之一，表示与检测条件相匹配。

例如：CASE 12,14,16,18,20,22
    CASE “ABC”,“NAJING”

② <常量 1> TO <常量 2>(或<常量 1>:<常量 2>)

<表达式>的值在常量范围中，表示与检测条件相匹配。

例如：CASE 100 TO 200
    CASE“MON” TO “SAT”

③ IS <关系运算符> <常量>

<表达式> 的值与常量满足关系运算符所表达的关系，则表示它与检测条件相匹配。上述三种形式的测试条件，可以在一个 CASE 语句中综合使用。

例如：CASE －65 TO 65，75，IS > 200

（4）所有检测条件必须都不相同，即不能同时选择执行两个不同的语句块。

例 4.12　输入一个月份数字，转换为一个英文月份名称输出。

```
! Convert month value to English month neme
INPUT prompt"Enter month(1—12):";m
SELECT CASE m
  CASE 1
    PRINT "MONTH no. ";m;"is January"
  CASE 2
    PRINT "MONTH no. ";m;"is February"
  CASE 3
    PRINT "MONTH no. ";m;"is March"
  CASE 4
    PRINT "MONTH no. ";m;"is April"
  CASE 5
    PRINT "MONTH no. ";m;"is May"
  CASE 6
    PRINT "MONTH no. ";m;"is June"
  CASE 7
    PRINT "MONTH no. ";m;"is July"
  CASE 8
    PRINT "MONTH no. ";m;"is August"
  CASE 9
    PRINT "MONTH no. ";m;"is September"
  CASE 10
    PRINT "MONTH no. ";m;"is October"
  CASE 11
    PRINT "MONTH no. ";m;"is November"
  CASE 12
    PRINT "MONTH no. ";m;"is December"
  CASE else
    PRINT "WRONG input data"
END SELECT
END
```

……………………

```
ok. run
Enter month(1—12);5
Month No.  5 is May
ok. run
Enter month(1—12);14
Wrong input data
ok.
```

例 4.13　由键盘输入一个学生的某科成绩 X，按下表要求打印等级：

成绩 X	<60	60～69	70～79	80～89	90～99	100
等　级	DOWN	PASS	C	B	A	EXCELLER

```
! print the score of students
INPUT prompt"enter the score of a student ?":x
SELECT CASE x
    CASE 0 to 59
      PRINT "DOWN"
    CASE 60 to 69
      PRINT "PASS"
    CASE 70 to 79
      PRINT "C"
    CASE 80 to 89
      PRINT "B"
    CASE 90 to 99
      PRINT "A"
    CASE 100
      PRINT "EXCELLER"
    CASE else
      PRINT "WRONG input data"
END SELECT
END
```

……………………

```
ok. run
Enter the score of a student 68
PASS
ok. run
Enter the score of a student 56
DOWN
```

```
ok. run                                  A
Enter the score of a student 98          ok.
```

4.4 计数循环结构

判断选择结构虽然能够有效地实现程序的条件转移，但却无法控制某一程序段的重复执行。在程序设计中，常常需要由一定的条件控制，反复执行某个程序段，完成这种功能的程序结构称为循环结构。True BASIC 提供了 FOR－NEXT 循环（计数循环）和 DO－LOOP 循环（条件循环）。

4.4.1 FOR－NEXT 循环结构

FOR－NEXT 是计数型循环结构，一般是在事先知道循环次数的情况下使用，它的基本特点是由一个循环变量计数控制循环，每循环一次，循环变量的值被修改一次，当循环次数已够数时，则结束循环。

```
格式： FOR <循环变量>=<初值> TO <终值> STEP <步长值>
              ..................  ⎫
              ……语句块……          ⎬循环体
              ..................  ⎭
          NEXT <循环变量>
```

其中，<循环变量> 可以用任一简单变量表示，它是控制循环执行次数的变量，所以只可能是数值型变量；<初值>、<终值> 及 <步长值> 为数值表达式，可以是常数，或已赋过值的数值变量或数值表达式。<步长值>是指循环每重复一次，循环变量的修改值。

FOR 是循环结构的起始语句，NEXT 为循环结构的终止语句，二者必须成对出现，缺一不可。FOR 后面的循环变量名与 NEXT 后面的循环变量名应一致。

FOR 与 NEXT 之间的语句块称为循环体，循环体中可以是一条或多条可执行语句，或是选择结构和另一个循环结构。如循环体中无任何语句存在时，则构成空循环，只消耗循环起始语句与循环终止语句的执行时间而不作任何具体操作，通常作定时的延时使用。

功能：按预定次数执行循环体。

程序执行进入循环结构时，循环结构的 FOR 语句有两个作用：

（1）初值赋于循环变量，确定<初值>、<终值>、<步长值> 并存贮起来。

（2）把循环控制变量与终值比较，当循环变量的值越过终值时，则突破循环结构，控制转向 NEXT 语句下一行语句执行；否则执行循环体语句。NEXT 语句亦有两个作用：① 修改循环变量的值，把循环变量的当前值加上步长值再赋给循环变量；② 无条件转向 FOR 语句，此时，只执行 FOR 语句的第二个作用。

循环变量的值按步长修改，循环一次，修改一次，因此 FOR－NEXT 循环结构又称计数循环结构。

说明：

（1）程序进入循环结构时，初值、终值和步长值由 FOR 语句设定，循环变量控制的循环次数按下列公式计算：

循环次数＝INT((终值－初值)/步长值)＋1

因此，一旦进入 FOR—NEXT 循环，循环次数就确定。

例 4.14　本例循环体中虽改变了作为初值、终值、步长值的变量 a、b、c 的值，但不影响循环的次数。

```
! Observe the numbers of the circulate.
LET a=1
LET b=10
LET c=2
PRINT"i","a","b","c"
FOR i=a to b step c
    PRINT i,a,b,c
    LET a=0
    LET b=0
    LET c=1
NEXT i
END
```

```
........................
ok. run
i    a    b    c
1    1    10   2
3    0    0    1
5    0    0    1
7    0    0    1
9    0    0    1
ok.
```

(2) 步长值可以是正数，也可以是负数。当步长值为正时，循环控制变量的初值必须小于终值，否则不执行循环，且循环控制变量越过终值的含义是大于终值。当步长值为 1 时，STEP 1 可省略。当步长值为负时，循环控制变量的初值必大于终值，否则不执行循环，且循环控制变量越过终值的含义是小于终值。

例 4.15　打印 10 到 25 之间的偶数。

```
! print even numbers between 10 to 25
FOR k=10 to 25 step 2
    PRINT k;
NEXT k
END
```

```
........................
ok. run
 10 12 14 16 18 20 22 24
ok.
```

例 4.16　打印 25 到 10 之间的奇数。

```
! print odd numbers from 25 down to 10
FOR k=25 to 10 step -2
    PRINT k;
NEXT k
END
```

```
........................
ok. run
 25 23 21 19 17 15 13 11
ok.
```

例 4.17　打印 1 到 10 之间整数的平方、立方和 4 次方表。

```
! print the square and cube and quartic for k
PRINT "k","^ 2","^ 3","^ 4"
FOR k=1 to 10
    PRINT k,k^ 2,k^ 3,k^ 4
NEXT k
END
```

```
........................
ok. run
k        ^ 2      ^ 3      ^ 4
 1        1        1        1
2        4        8        16
3        9        27       81
4        16       64       256
5        25       125      625
6        36       216      1296
7        49       343      2401
8        64       512      4096
9        81       729      6561
10       100      1000     10000
ok.
```

(3) 循环变量在循环体中不被引用，循环变量的作用仅仅控制循环次数；循环变量在

循环体中被引用，这时它不仅控制循环体执行次数，由于它随循环进行不断修改，循环变量的值在每次循环中不同，这在程序设计中十分有用。例 4.15，例 4.16，例 4.17 都是循环变量在循环体中被引用的例子。

例 4.18 计算一个句子中的元音字母和辅音字母的个数，及它们在该句子中所占的百分数。

解 英语单词中有 a，e，i，o，u 共有五个元音，其余的均是辅音。一个句子由英语单词组成，各单词之间需用空格隔开，因此程序中要累计出空格的个数。一个句子的长度可由 LEN（A $）函数决定，去掉元音和空格外则是辅音字母的个数。

```
! print the numbers of the vowel and the consonant
LET a,e,i,o,u,k,f=0
LINE INPUT a $
FOR p=1 to len(a $)
  SELECT CASE a $[p:p]
    CASE "a","A"
      LET a=a+1
    CASE "e","E"
      LET E=E+1
    CASE "I","i"
      LET i=i+1
    CASE "o","O"
      LET O=O+1
    CASE "U","u"
      LET u=u+1
    CASE “”
      LET k=k+1
    CASE else
      LET f=f+1
  END SELECT
NEXT p
PRINT "a=";a,"e=";e,"i=";i,"o=";o,
PRINT "u=";u
PRINT "V-percent=";
PRINT (A+E+I+O+U)/ LEN(A $) * 100;"%"
PRINT "F=";F
PRINT "C-percent=";
 PRINT F/LEN(A $) * 100;"%"
END
……………………
ok. run
? screen prompt
a=0     e=2     i=0     o=1     u=0
V-percent= 23.0769%
f= 9
C-percent= 69.2308%
ok.
```

例 4.19 用梯形法求定积分 $\int_a^b \sin x dx$

解 求 f(x)在[a，b]区间上的定积分（见图 4.1），它的几何意义是求曲线 y=f(x)和 x=a，y=0，x=b 所围成的曲面梯形的面积。若把[a，b]等分成 N 个区间，每个区间的长度为：

$$h=(b-a)/n$$

从每个等分点上作 x 轴的垂线，将所求面积分成 n 个小曲边梯形。每个曲边梯形的面积近似等于：$(f(x_i)+f(x_i+h))*h/2$，其中 $x_i=a+ih$。

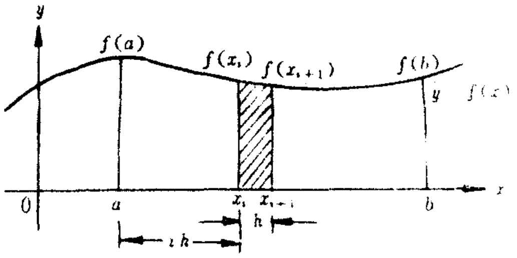


图 4.1 梯形法求定积分

将上述 n 个小梯形的面积相加，可得到所求曲边梯形面积的近似值。n 越大，求出的值越接近积分的真值。

```
! Trapezoidal method
INPUT prompt "a,b,n=";a,b,n
```

```
LET h=(b-a)/n
LET x=a
LET f0=sin(x)
FOR i=1 to n
    LET x=x+h
    LET f1=sin(x)
    LET t=(f0+f1)*h/2
    LET s=s+t
    LET f0=f1
NEXT i
PRINT"a=";a,"b=";b,"n=";n
PRINT using "s=###.########";s
END
........................
ok. run
a,b,n= 0,3.1416,100
a= 0      b= 3.1416      n= 100
s=1.99983550
ok. run
a,b,n= 0,3.1416,1000
a= 0      b= 3.1416      n= 1000
s= 1.99999836
ok.
```

4.4.2 非正常出口 EXIT FOR 语句

FOR－NEXT 结构的循环次数由 FOR 语句中的三个参数:初值、终值及步长值决定,适用于循环次数已知的情况,但 FOR－NEXT 结构亦可用于解决循环执行次数不十分明确的场合,例如许多代数题求未知数。计算程序并不能直接求解未知数,但可预估未知数的可能范围,利用穷举试凑方法求解,此时可能出现:往往在满足解题要求后,而循环变量没有达到终值的现象,如果没有一定措施,这时还要做许多无效的循环,只能到循环变量越过终值,正常地退出循环结构。为避免浪费时间,True BASIC 提供了一种非正常出口的手段,即 EXIT FOR 语句。

格式: EXIT FOR

功能: 无条件跳出所在的 FOR 循环结构, 程序转移到 NEXT 语句的后继语句继续执行。

说明:EXIT FOR 语句只允许在 FOR－NEXT 循环结构的循环体内使用,不允许在循环体外使用。实际上独立使用 EXIT FOR 毫无意义,因为非正常出口总应该有理由,满足某个给定条件时才用。因此,EXIT FOR 语句通常与 IF 语句一起使用。

例 4.20 鸡和兔同笼共 50 只,总的脚数为 120,试问鸡和兔各多少只?

解 因不知道兔子有多少只,但至少有一只,再按脚数分析,最多不超过 30 只,这样选择 I 为循环控制变量代表兔子数,则鸡为 J=50－I,采用穷举算法,I 的初值为 1,终值为 30,步长为 1。由题意可知,退出循环的条件是:

$$4*I+2*J=120$$

```
! COUNTER
FOR i=1 to 30
    LET j=50-i
    IF 4*i+2*j=120 then EXIT FOR
NEXT i
IF i<=30 then
    PRINT "i=";i,"j=";j
ELSE
    PRINT "no-result."
END IF
END
....................
ok. run
i= 10          j= 40
ok.
```

4.4.3 计数循环结构的应用实例

例 4.21　求 S=1+2+3+……+50

解　变量 S 存放 1～50 个数的累加和，一般存放累加和变量的初值为 0，初值赋值语句应放置在循环结构之外，FOR 语句之前。设循环控制变量为 I，其初值为 1，终值为 50，步长为 1，循环 50 次。I 既控制循环的次数，又是每次循环中被累加的数，循环体中用赋值语句：

```
LET     S      =      S +I
      （新值）     （旧值）
```

通过迭代来实现 1～50 个数的累加。

开始，S 初值为 0，第一次循环时 I=1，当执行完第一次循环后，S 中得到新值 1。第二次循环时，S 中旧值为 1，I=2，当执行完第二次循环后，S 中得到新值为 3。这样，依次类推，循环 50 次后，即可求得 50 个数之和。

```
! print the sum of fifty numbers from 1 to 50
LET s=0
FOR i=1 to 50
LET s=s+i
NEXT i
PRINT ″S=″;S
END
.......................
ok. run
S= 1275
ok.
```

例 4.22　求 n! =1*2*3*......*(n−1)*n

解　求阶乘，即求连乘积，一般存放连乘积的变量的初始值为 1，赋初值应放置在循环结构之外，FOR 语句之前。本例也用迭代算法。

```
! PRINT the factorial of the n
INPUT prompt″Enter the number″;n
LET k=1
FOR i=1 to n
  LET k=k * i
NEXT i
PRINT″THE factorial of the″;n;″is″;k
END
.......................
ok. run
Enter the number15
THE factorial of the 15 is 1.30767e+12
ok.
```

例 4.23　已知自然数 e 可用下述级数表示：

$$e=1+1/1!\ +2/2!\ +1/3!\ +\cdots+1/n!\ +\cdots$$

设计一个程序，求 e 的近似值(设 n=10)。

```
! print an approximate value of e
LET e=1
LET k=1
FOR n=1 to 10
    LET k=k * n
    LET e=e+1/k
NEXT n
PRINT ″e=″;e
END
.......................
ok. run
e= 2.71828
ok.
```

例 4.24　某部战士 663 人行军到河边，需乘船到对岸。从附近 5 个村子借来一批小船，每船运载若干人，且各船运载的人数相等，结果只用了 3 次往返就把战士全部送到对岸。问借了多少船？每船运载多少人？

解　采用双重循环，穷举算法。

外循环控制变量为 X，用来选择船的数目。船是从 5 个村子借来的故不能小于 5，取 X 初值为 5。因为 3 次运完，每次每船至少 2 人，所以船数不可能大于 663/(2 * 3)，取整则 X 终值取 110，增量为 1。

内循环控制变量 Y，用来决定每船运载的人数。每船至少为 2 人，故 Y 初值取 2。因船只最少为 5 只，3 次运完，因此每船最大运载量为 663/(3 * 5)，取整则 Y 的终值为 44，增量为 1。列举所有情况，当满足条件：3 * X * Y＝663，退出循环，求得船数和每船载人数。

```
! print the numbers of the boats and the persona
FOR x=5 to 110
    FOR y=2 to 44
      IF 3 * x * y=663 then
        PRINT "x=";x,"y=";y
        EXIT FOR
     END IF
    NEXT y
    IF y<=44 then EXIT FOR
NEXT x
END
........................
ok. run
x= 13      y= 17
ok.
```

例 4.25　用牛顿切线法求代数方程 $e^{-x}-x=0$ 在区间 (0,1) 上的一个近似根，算法的几何意义如图 4.2 所示，假设 x_1 是所求之近似根，那么曲线在 x_1 点处的导数为：

$$f'(x_1)=\frac{f(x_1)}{x_1-x_2}\qquad x_2=x_1-\frac{f(x_1)}{f'(x_1)}$$

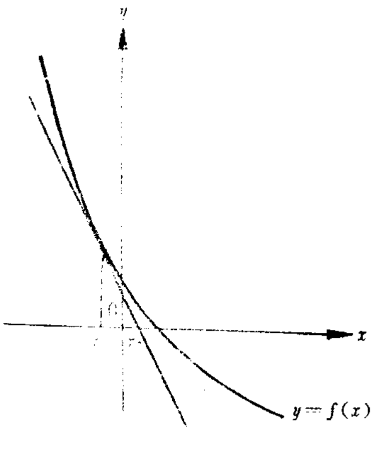


图 4.2　牛顿迭代法示意图

由 x_1 点可求得 x_2 点，x_2 是上述切线与 x 轴之交点，x_2 比 x_1 更接近于真根。用同样的方法，由 X_2 可求得 $X_3\cdots$，这样逐次迭代，X 便逼近方程的真根。判别是否达到精度要求，可以用前后两次求得的近似根之差的绝对值 $|X_{i+1}-X_i|$ 是否小于给定允许的误差 ε。如满足条件 $|x_{i+1}-x_i|<=\varepsilon$，则迭代完成，根已求得。

牛顿迭代公式为：

$$x_{i+1}=x_i\frac{f(x_i)}{f'(x_i)}$$

本例中：$f(x)=e^{-x}-x$，$f'(x)=-e^{-x}-1$，取初值 $x_1=1$，允许误差＝1e－8。

```
! Newton' s method of iteration
INPUT prompt "x=";x
```

```
PRINT "n","x","difference"
FOR i=1 to 200
  LET n=n+1
  LET x1=x
  LET f=exp(-x1)-x1
  LET f1=-exp(-x1)-1
  LET x=x1-f/f1
  LET d=abs(x-x1)
  PRINT n,
  PRINT using"##.##########";x,d
  IF d<=1e-8 then EXIT FOR
NEXT i
END
……………………
ok. run
x=1
n         x                difference
  1       .5378828427      .4621171573
  2       .5669869914      .0291041487
  3       .5671432860      .0001562946
  4       .5671432904      .0000000044
ok.
```

牛顿迭代法的收敛性依赖于初值 x_1 的选取，如 x_1 偏离所求根较远，则迭代过程可能发散，即输出的 difference 项值可能越来越大。当发散时，应重新选取初值 x_1，进行计算。

4.5 条件循环结构

计数循环结构 FOR－NEXT 的特点是：重复执行循环体的次数由一个循环变量控制，一般用于循环次数已知的情况。在实际应用中，许多问题在求解过程中预先不知道重复执行循环体的次数，为此，True BASIC 提供了另一种条件循环 DO－LOOP 结构。在 DO 循环中使用 WHILE<条件>短语或 UNTIL<条件>短语控制循环。

4.5.1 条件循环 DO－LOOP 结构

DO 循环的一般格式：

```
DO [WHILE <条件 1> / UNTIL <条件 2> ]
    ………………  ┐
    ……语句块……  ├循环体
    ………………  ┘
LOOP [WHILE <条件 3> / UNTIL <条件 4> ]
```

方括号表示其中的内容是可以任选的。这样，DO 循环的可能形式有下列九种：

```
① DO WHILE <条件>
    ………………  ┐
    ……语句块……  ├循环体
    ………………  ┘
LOOP
② DO
    ………………  ┐
    ……语句块……  ├循环体
    ………………  ┘
LOOP WHILE <条件>
③ DO UNTIL <条件>
    ………………  ┐
    ……语句块……  ├循环体
    ………………  ┘
LOOP
④ DO
    ………………  ┐
    ……语句块……  ├循环体
    ………………  ┘
LOOP UNTIL <条件>
⑤ DO WHILE <条件 1>
```

```
        ………………
        ……语句块……  }循环体
        ………………
    LOOP WHILE <条件 2>
⑥ DO UNTIL <条件 1>
        ………………
        ……语句块……  }循环体
        ………………
    LOOP UNTIL <条件 2>
⑦ DO WHILE <条件 1>
        ………………
        ……语句块……  }循环体
        ………………
    LOOP UNTIL <条件 2>
⑧ DO UNTIL <条件 1>
        ………………
        ……语句块……  }循环体
        ………………
    LOOP WHILE <条件 2>
⑨ DO
        ………………
        ……语句块……  }循环体
        ………………
    LOOP
```

功能：根据给定条件重复执行循环体。循环结构执行过程如下：

① 先检查 DO 部分的条件。如 DO 后面无条件短语，则无条件地执行循环体；当 DO WHILE 后的条件为真或 DO UNTIL 后的条件为假时，则执行循环体；否则跳出循环结构，执行 LOOP 的后继语句。

② 执行循环体后，检查 LOOP 部分的条件。如 LOOP 后未跟条件短语，或 LOOP WHILE 后的条件为真或 LOOP UNTIL 后的条件为假，则控制转向 DO 语句；否则突破循环，执行 LOOP 的后继语句。DO 和 LOOP 后面如都无条件限制，将无休止地循环，称死循环，除非在循环体中有非正常出口 EXIT DO 语句。

说明：

① <条件>可以是关系表达式、逻辑表达式或逻辑函数。条件短语在 DO 之后，循环体有可能一次都不执行。条件短语在 LOOP 之后，循环体至少执行 1 次。

② DO－LOOP 结构有两个条件控制突破循环的正常出口，两个出口的会合点是 LOOP 的后继语句。

③ 循环条件可用 WHILE 短语，亦可用 UNTIL 短语，或二者都用。

(1) WHILE 短语

在 DO－LOOP 结构中，DO 语句或 LOOP 语句中均可使用 WHILE <条件>短语控制循环。无论 WHILE<条件>短语用于循环结构的 DO 语句或 LOOP 语句(见格式①、②结构)，只有条件为真时才进行循环。结构①是在 DO 中先判断条件，如果条件不成立，立即突破循环结构，循环一次也不执行；否则执行循环体。结构②是直接进入循环，然后在 LOOP 中由 WHILE 短语判断条件是否成立，若成立再执行循环，若条件不成立突破循环结构，执行 LOOP 后继语句。

例 4.26 观察 WHILE 短语

方案 1：WHILE 短语在 DO 后面，先判断条件，因条件不成立，循环一次也没有执行，所以 S＝0

```
! Observe while phrase
LET s=0
```

方案 2：WHILE 短语在 LOOP 后面，所以不管条件成立与否，至少要执行一次循环。

```
! Observe while phrase
LET s=0
LET b=10
LET a=28
DO while b>a
  READ a
  PRINT a;
  LET s=s+a
LOOP
PRINT
PRINT"s=";s
DATA 2,4,6,8,10,12,14
END
........................
ok. run
S= 0
ok.
```

```
LET b=10
LET a=28
DO
  READ a
  PRINT a;
  LET s=s+a
LOOP while b>a
PRINT
PRINT"s=";s
DATA 2,4,6,8,10,12,14
END
........................
ok. run
  2   4   6   8   10
s= 30
ok.
```

(2) UNTIL 短语

在 DO 循环中，DO 语句和 LOOP 语句均可使用 UNTIL<条件>短语控制循环（见格式③、④结构），无论结构③和结构④的共同点是只有当 UNTIL 给出的条件为假时才进行循环。结构③是在 DO 中由 UNTIL 短语先判断条件，如果条件为假时进入循环，直到条件为真时结束循环。结构④是无条件地进入循环，然后在 LOOP 中由 UNTIL 短语判断条件，如果条件为假时继续循环，直到条件为真时结束循环。

例 4.27　假定在银行中存款 10000 元，按 11.25%的年利息计算，试问过多少年就会连本带利翻一翻。

方案 1：DO UNTIL <条件>

```
! compute the interest
LET m=10000
LET orig=m
LET inte=11.25/100
PRINT "year","money"
DO until m>2 * orig
  LET year=year+1
  LET m=m * (1+inte)
  PRINT year,m
LOOP
PRINT year;"years";"have";m
END
........................
ok. run

  year          mony
    1             11125
```

方案 2：LOOP UNTIL <条件>

```
! compute the interest
LET m=10000
LET orig=m
LET inte=11.25/100
PRINT "year","money"
DO
LET year=year+1
  LET m=m * (1+inte)
  PRINT year,m
  LOOP until m>=2 * orig
PRINT year;"years";"have";m
END
........................
ok. run

  year          mony
    1             11125
```

```
2                  12376.6
3                  13768.9
4                  15317.9
5                  17041.2
6                  18958.3
7                  21091.1
7 years have 21091.1
ok.
```

```
2                  12376.6
3                  13768.9
4                  15317.9
5                  17041.2
6                  18958.3
7                  21091.1
7 years have 21091.1
ok.
```

方案 1 和方案 2 程序运行结果是相同的。

(3) 各种 DO－LOOP 条件循环结构

各种 DO－LOOP 结构实际是由 WHILE＜条件 1＞与 UNTIL＜条件 2＞各种组合的结果。现在我们来分析其中一种，其它的组合结构也就不难明白了。

```
DO WHILE <条件 1>
     ……………………………… }
     ……语句块……         }循环体
     ……………………………… }
LOOP UNTIL <条件 2>
```

这种 DO－LOOP 结构，先测试 WHILE＜条件 1＞，当＜条件 1＞为真时则进入循环，到 LOOP 处又测试 UNTIL＜条件 2＞，当＜条件 2＞不成立则返回循环起点再测试＜条件 1＞。

例 4.28　某设计工作中有计算公式：

$$S=\begin{cases} 17000-0.485R^2 & (R<120) \\ \dfrac{18000}{1+\dfrac{R^2}{18000}} & (R>=120) \end{cases}$$

要求对参数 R 从 20 到 200 的范围内，每隔 50 计算一次 S。

程序 1：DO WHILE ＜条件 1＞ 与 LOOP UNTIL ＜条件 2＞ 结构程序。

```
! print the value of the s
PRINT "r","s"
LET r=20
DO while r>=20
  IF r<120 then
    LET s=17000-0.485 * r * r
  ELSE
     LET s=18000/(1+r * r/18000)
   END IF
   PRINT r,s
   LET r=r+50
 LOOP until r>200
 END
```

程序 3：DO WHILE ＜条件 1＞与 LOOP UNTIL＜条件 2＞结构程序。

程序 2：DO UNTIL ＜条件 1＞与 LOOP WHILE ＜条件 2＞ 结构程序。

```
! print the value of the s
PRINT"r","s"
LET r=20
DO until r<20
  IF r<120 then
    LET s=17000-0.485 * r * r
  ELSE
    LET s=18000/(1+r * r/18000)
   END IF
  PRINT r,s
  LET r=r+50
LOOP while r<=200
END
```

程序 4：DO UNTIL ＜条件 1＞与 LOOP WHILE ＜条件 2＞ 结构程序。

```
! print the value of the s
PRINT"r","s"
LET r=20
DO while r>=20
  IF r<120 then
    LET s=17000-0.485*r*r
  ELSE
    LET s=18000/(1+r*r/18000)
  END IF
  PRINT r,s
  LET r=r+50
LOOP while r<=200
END
.................
ok. run

r               s
  20            17608.7
  70            14148.5
  120           10000.
  170           6908.32
ok.
```

```
! print the value of the s
PRINT"r","s"
LET r=20
DO until r<20
  IF r<120 then
    LET s=17000-0.485*r*r
  ELSE
    LET s=18000/(1+r*r/18000)
  END IF
  PRINT r,s
  LET r=r+50
LOOP until r>200
END
.................
ok. run

r               s
  20            17608.7
  70            14148.5
  120           10000.
  170           6908.32
```

以上四个程序的执行结果是一样的。

(4) DO—LOOP 结构特点

① DO—LOOP 条件循环结构中的 <条件>可放在 DO 后面，也可放在 LOOP 后面。条件短语有 WHILE(又称当型条件)和 UNTIL(又称直到型条件)，也可以是它们的各种组合。因此可组成各种 DO—LOOP 循环结构，使用十分方便灵活。

② DO—LOOP 循环结构的正确使用：循环控制条件初值和循环中所处理变量的初值应在结构外先设置，否则会形成死循环或处理出错；循环体中应有改变循环控制条件的语句。在经过有限次循环之后，循环的测试条件就能够从成立到不成立(或者相反)。控制跳出循环体，否则也会造成死循环。

③ 正确设置循环条件非常重要，一般是由所给问题及对该问题进行处理所采用的算法决定。

例 4.29　使用格利高利公式求 π 的值：

$$\pi=4[1-1/3+1/5-1/7+1/9-1/11+\cdots]$$

解　由公式可知，π 值是一个收敛的无穷级数。实际计算 π 的近似值，只要满足一定精度要求，例如某项绝对值小于 1E—5。

```
! compute the value of pi
LET pi0=0
LET t,r,s=1
DO while abs(t)>1e-5
```

```
LET pi0=pi0+t
LET r=r+2
LET s=-s
LET t=s/r
```

```
LOOP                                   ........................
LET pi0=pi0 * 4                        ok. run
PRINT "PI=";PI0                        pi= 3.14157
END                                    ok.
```

终值控制：终值只用作循环终止条件而设置，本身对程序处理无直接关系。本例的终值是常数 1E−5。若条件放在 DO 后面时为 DO WHILE ABS(T)>1E− 5，放在 LOOP 后面时改为 LOOP UNTIL ABS(T)<1E−5。

例 4.30　从键盘输入一批数，求其正数之和，当输入的数为−1 时，表示数据输入完毕。

```
! Use special data as the end          ........................
LET s=0                                ok. run
DO                                     ? 5
    INPUT x                            ? 7
    IF x>0 then LET s=s+x              ? 8
LOOP until x=−1                        ? −1
PRINT "s=";s                           s=20
END                                    ok
```

采用一种特殊的结束标志来终止循环，尤其在数据处理中，通常可以使用 INPUT 语句键入，如−1，9999 等特殊的数值作为结束标志，它具有直观和方便的优点。

4.5.2　EXIT DO 语句

格式：EXIT DO

功能：跳出它所在的 DO−LOOP 循环结构，执行 LOOP 语句的后继语句。EXIT DO 语句必须在 DO−LOOP 循环体内使用。它与 EXIT FOR 语句类似，通常与 IF 语句一起使用，当满足一定条件时跳出循环。此语句不允许在循环体外使用。在循环嵌套的情况下，它只能跳出所在层的循环结构。

无条件短语的 DO−LOOP 循环结构(格式⑨)，它是一个无限循环，有时也称为"死循环"，必须利用 EXIT DO 语句才能从循环体中跳出来。

例 4.31　在 0～999 范围内，找出其值等于该数中各位数字的立方和的数。

解　利用穷举算法，从 0 开始检测，打印满足条件的数，否则数值增 1 修改成下一个数，逐个循环检测，直到数大于 999 时，利用 EXIT DO 语句脱离无条件短语的 DO−LOOP 循环。

利用取整函数 INT(N) 取出各位的值：

百位数 A=INT(N/100)

十位数 B=INT((N−100 * A)/10)

个位数 C=N−100 * A−10 * B

当满足条件 $N=A^3+B^3+C^3$ 时输出结果。

```
! Use the exit do
LET n=0
DO
  IF n>999 then EXIT DO
LET a=int(n/100)
LET b=int((n−100 * a)/10)
LET c=n−100 * a−10 * b
IF n=a^ 3+b^ 3+c^ 3 then
```

```
    PRINT "a=";a,"b=";b,"c=";c,
    PRINT "n=";n
  END IF
  LET n=n+1
LOOP
END
……………………
ok. run
```

```
a= 0    b= 0    c= 0    n= 0
a= 0    b= 0    c= 1    n= 1
a= 1    b= 5    c= 3    n= 153
a= 3    b= 7    c= 0    n= 370
a= 3    b= 7    c= 1    n= 371
a= 4    b= 0    c= 7    n=407
ok.
```

4.5.3 逻辑函数

READ/DATA 语句，DATA 语句应该保证供给 READ 语句读入足够的数据，否则将出现错误信息，为检查 DATA 语句中是否还有数据可读，在 True BASIC 中提供了两个数据测试函数 MORE DATA 和 END DATA，另外还提供了一键盘输入测试函数 KEY INPUT。

(1) 有数据读入测试函数(MORE DATA)

格式：MORE DATA

功能：若 DATA 语句中有数据项可供 READ 语句读入，则 MORE DATA 逻辑函数值为“真”，否则为“假”

(2) 无数据读入测试函数(END DATA)

格式：END DATA

功能：它与 MORE DATA 函数正好相反，当 DATA 语句中无数据项可供 READ 语句读入时，其逻辑值为“真”，否则为“假”。

例 4.32　求数据表中若干个数之和。

方案 1：MORE DATA 函数

```
! print the sum
LET sum=0
DO while more data
   READ n
   LET sum=sum+n
LOOP
PRINT “sum=”;sum
DATA 11,12,34,56,78,89,93,34,28,58
END
ok. run
sum= 493
ok.
```

方案 2：END DATA 函数

```
! print the sum
LET sum=0.
DO until end data
   READ n
   LET sum=sum+n
LOOP
PRINT “sum=”;sum
DATA 11,12,34,56,78,89,93,34,28,58
END
ok. run
sum= 493
ok.
```

(3) 键盘输入测试函数(KEY INTUT)

格式：KEY INPUT

功能：检查是否按了一下键。若按任一键，则其值为“真”，否则为“假”。

说明：执行 INPUT 和 GET KEY 语句时，程序都要暂停，等待按键输入数据，然而有时要求没有键按下时，程序去做某些事，只是在有键按下时才作键处理，这时就要用 KEY INPUT 函数来实现这个功能。

例 4.33　未按键时屏幕上显示数值不断增加，按任意一个键后停止。

```
! key input test
PRINT “press any key to stop”
DO until key input
    LET i=i+1
    PRINT i,
LOOP
END
························
ok. run
```

```
press any key to stop
  1       2       3       4       5
  6       7       8       9      10
 11      12      13      14      15
 16      17      18      19      20
 21      22      23      24      25
 26      27      28      29      30
ok
```

逻辑函数是不带自变量的。

综上所述，逻辑条件可用关系表达式、逻辑表达式或逻辑函数来表示，它们最后所取的结果是“真”或“假”。因此程序中需进行逻辑判断的地方，均可以设置关系表达式、逻辑表达式或逻辑函数，根据其结果的“真”或“假”来控制程序的流程。

例 4.34　一个数列由 160 个数组成，头两个数为 0，1，随后的各数依次是前两个数之和，编写程序，输出该数列。

解　本题关键是如何生成该数列后 158 个元素，据题意可知，若要知道 K 元素必须知道和保留 K－2，K－1 两个元素的值，故可用两个存储单元 A，B 存放 K－2，K－1 的值，一旦生成 K，则 K－2 就可让出给 K 保存数值。因此，

$$\underset{(K-2)}{A} + \underset{(K-1)}{B} \longrightarrow \underset{(K)}{A} \qquad \underset{(K-1)}{B} + \underset{(K)}{A} \longrightarrow \underset{(K+1)}{B}$$

称为递推算法，即从前面一些量可以依次推出后面一些量的算法。递推常常表现为“迭代”形式，即有一个变量的原值推出它的新值，如上面分析中的 $A+B\longrightarrow A$，$B+A\longrightarrow B$，采用循环重复执行这两条语句，可获得结果。常常把这种新值与旧值的关系式称迭代公式，而 A，B 称迭代变量，这种算法称迭代算法。

输出 160 个数的数列过于冗长，为使程序按所需中止，设置 KEY INPUT 函数作非正常出口。

```
! print the enumeration
LET a=0
LET b=1
PRINT “a”,“b”
PRINT a,b
LET i=2
DO while i<=80
    LET a=a+b
    LET b=b+a
    PRINT a,b
    LET I=I+1
    IF KET INPUT then EXIT DO
LOOP
PRINT
END
························
ok. run
```

```
a              b
  0              1
  1              2
  3              5
  8             13
 21             34
 55             89
 144            233
 377            610
 987            1597
 2584           4181
```

```
6765            10046            46368            75025
17711           28657            ok.
```

注：因使用了 KEY INPUT 函数，按任一键可中止程序运行。

4.5.4 条件循环结构的应用实例

例 4.35 试从一批数据（包括学生姓名及考试成绩）中，统计出不及格学生的姓名及最高的成绩。数据由 DATA 语句给出。

程序及运行结果如下：

```
! Statistical work
PRINT “no-passed students:”;
LET high=0
DO while more data
    READ name $ ,score
    IF score<60 then PRINT name $ ;“   ”
       ;else LET high=max(high,score)
LOOP
PRINT
PRINT “Largest score:”;high
DATA Zhang,80,Li,75,Wang,45,Zhao,65,
DATA Chen,58
DATA Guo,95,Lin,58,Ma,54,Fang,89,Gan,  91
END
......................
ok. run
no-passed students:Wang   Chen   Lin   Ma
Largest score: 95
ok.
```

例 4.36 求阶乘的和 $M=\sum_{N=1}^{20} N!$

解 本题的展开式：$M=1!+2!+3!+\cdots+20!$，可用迭代算法求解。

```
! print the sum of the factorial
LET m=0
LET t,n=1
DO while n<=20
    LET t=t * n
    LET m=m+t
    LET n=n+1
LOOP
PRINT “m=”;m
END
......................
ok. run
m= 2.56133e+18
ok.
```

例 4.37 求最大公约数。编一个程序求 N 个自然数的最大公约数。

解 N 个自然数的最大公约数必定是这 N 个自然数中最小数的约数。算法如下：

(1)先求 N 个数中的最小数 MIN。

(2)用 2—MIN 分别去除各数，都能整除者，即是最大公约数。

```
! the maximum common divisor
LET n=3
READ n1,n2,n3
PRINT “(”;n1;“,”;n2;“,”;n3;“)=”;
LET S=n1
LET S=min(min(n1,n2),n3)
LET t=0
LET i=S
DO while t=0 and i>=2
    IF mod(n1,i)=0 and mod(n2,i)=0 and
      mod(n3,i)=0 then
          PRINT i
          LET t=1
    END IF
    LET i=i-1
LOOP
IF t=0 then PRINT i
DATA 147,21,49
END
......................
ok. run
(147,21,49)= 7
```

```
ok.
```

例 4.38　相传古印度有个宰相叫达依尔，是国际象棋的发明者，非常聪明能干。国王舍罕准备奖励他，问他想要些什么？达依尔说：“国王只要在国际象棋的棋盘上的第一格子里放上 1 粒麦子，第二个格子里放 2 粒麦子，第三个格子里放 4 粒麦子，以后每个格子里加 1 倍，一直放完第 64 格，我就感恩不尽了。”国王想：“64 格有多少，这还不容易！”于是令人扛来一袋麦子，结果很快就用完了。又扛来一袋，很快又用完了。国王又命令扛几十袋麦子，结果按达依尔的要求还差得很远。国王非常奇怪，怎么需要这么多？

那么到底需要多少麦子，请编程求出。

解　64 个格子中应放的麦子粒数 T，可用下面公式算出：

$$T=2^0+2^1+2^2+2^3+\cdots+2^{63}$$

假定 1 立方米的麦子约有 1.42×10^8 粒，那么 T 粒麦子的体积为：

$$V=T/(1.42\times10^8)\text{立方米}$$

```
! counter the numbers and the volume of the
! wheats
LET t=1
LET b=1
LET n=1
DO while n<=63
     LET b=b * 2
     LET t=t+b
     LET n=n+1
LOOP
PRINT “t=”;t
LET v=t/(1.42 * 10^ 8)
PRINT “V=”;V
END
........................
ok. run
t= 1.84467e+19
v= 1.29907e+11
ok.
```

4.6　循环嵌套结构

在一个循环结构中又套有循环结构，称为循环嵌套结构。通常在处理一些比较简单的问题时，用单重循环结构便可解决，但当问题比较复杂时，必须用二重、三重或多重循环嵌套结构来解决。

例 4.39　打印九九乘法表就是含有二层循环嵌套结构。

```
! print the multiplicative table
set margin 80
set zonewidth 7
PRINT tab(20);“a * b multiplicative table”
PRINT
PRINT “a\b”,“1”,“2”,“3”,“4”,“5”,“6,”,
     “7”,“8”,“9”
FOR a=1 to 9
     PRINT a,
     FOR b=1 to 9
          PRINT a * b,
     NEXT b
     PRINT
NEXT a
PRINT
END
........................
ok. run
     a * b multiplicative table
a\b 1  2    3    4   5  6  7  8  9
1   1  2    3    4   5  6  7  8  9
2   2  4    6    8   10 12 14 16 18
3   3  6    9    12  15 18 21 24 27
4   4  8    12   16  20 24 28 32 36
```

```
5   5   10   15   20 25 30 35 40 45          8   8   16   24   32 40 48 56 64 72
6   6   12   18   24 30 36 42 48 54          9   9   18   27   36 45 54 63 72 81
7   7   14   21   28 35 42 49 56 63          ok.
```

例中有二重循环结构，外层结构循环变量为 A，内层结构循环变量为 B，外层循环每执行 1 次，内层循环要执行 9 次，然后 A 增加一个步长，B 再从 1 递增为 9，直到外循环执行完为止。

说明：

(1) FOR—NEXT 嵌套结构，各层循环变量不能同名。

(2) 在循环嵌套时，要层次清晰，无论是 FOR—NEXT 循环，还是 DO—LOOP 循环，均可自行嵌套，也可相互嵌套。如：

```
┌DO<条件 1>                ┌FOR X
│┌DO<条件 2>               │┌FOR Y
│└LOOP                     │└NEXT Y
│┌FOR X                    │┌DO <条件>
│└NEXT X                   │└LOOP
└LOOP                      └NEXT X
```

为便于查错、纠错，嵌套循环结构采用凹入写法，更显重要。

(3)循环允许嵌套，但不得交叉。下面是错误的情况：

```
 ┌FOR   X              ┌DO   <条件>
┌┼FOR   Y             ┌┼FOR   I
│└NEXT  X             │└LOOP
└─NEXT  Y             └─NEXT  I
```

(4)在某层循环如果有初值，不仅要注意合理选好初始值，而且一定要将变量赋初值语句设置在该层循环结构外，如果变量初值设置语句置于循环结构中，若是计算用变量，则计算结果出错，若是循环控制变量，则可能导致死循环。例如二重循环，内层循环中使用的变量或循环条件初值赋值语句应在内循环外，循环结构前；外循环的赋初值语句应设置在外循环外。

小　结

本章介绍了 True BASIC 程序设计中的判断选择结构和循环结构。

(1) 判断选择结构是判断给定的条件是否满足而决定程序的走向，其中条件是逻辑表达式、关系表达式以及 MORE DATA，END DATA 或 KEY INPUT 等逻辑函数。

True BASIC 提供的判断选择结构的形式有如下几种：

① 单行 IF—THEN 语句

```
IF<条件>THEN<语句 1>[ELSE<语句 2>]
```

② 多行 IF—TJEM 结构

```
IF<条件>THEN
  …语句块 1…
```

```
[ELSE
  …语句块 2…]
END IF
```

③ 多分支判断选择结构

```
IF<条件>THEN
  …语句块 1…
ELSE IF<条件 2>THEN
  …语句块 2…
ELSE IF<条件 3>THEN
  …语句块 3…
       ⋮
[ ELSE
  …语句块 N…]
END IF
```

④ SELECT CASE 选择语句

```
SELECT CASE <表达式>
      CASE <TEST11>,<TESE12>,……
          …语句块 1…
      CASE <TEST21>,<TEST22>,……
          …语句块 2…
                ⋮
      [CASE ELSE
          …语句块 N…]
END SELECT
```

(2) 循环结构是根据需要重复执行某一段程序。True BASIC 提供以下两种循环结构：

① 计数循环结构

```
FOR<循环变量>=<初值>TO<终值>STEP<步长>
    ……………………
    ……………………  }循环体
    ……………………
NEXT<循环变量>
```

② 条件循环结构

```
  DO[(WHILE <条件 1>/UNTIL <条件 2>]
      ……………………………
      ………语句块………  }循环体
      ……………………………
LOOP[(WHILE <条件 3>/UNTIL <条件 4>)]
```

(a) WHILE<条件>与 UNTIL<条件>可以省略或任意组合。

(b) FOR－NEXT 及 DO－LOOP 的循环内可含有 IF－THEN，IF－THEN－ELSE， SELECT CASE 等结构语句，并分别可利用 EXIT FOR，EXIT DO 语句跳出循环。

（c）两种循环结构可以互相嵌套。

习　　题

1. 指出下列语句中的错误并改正。

```
(1)if r≥5 then x=x+1 else x=x-1
```

```
(2)if r≤5
      then x=x+1
      else x=x-1
   end
```

```
(3)if a=b$ then
          print a,b$,c$
     else
     a=5;b$=ab;c$=a+b$
     if end
```

```
(4)select case x=a*b+c*d
     case x=2 or 3
          print "a"
     case x=4 and 5
          print "b"
     case x$="xyz","abc"
          print "type"
     else
          print "end"
     end case
```

2. 已知 p1$＝"Shaha"，p2$＝"Shanx"，p3$＝"Sichuan"，a＝5，b＝10，求下列逻辑表达式的值。

(1) p1$ >p2$ and p3$<p1$

(2) p1$< p3$ and not p2$<p3$

(3) 2*a <=b

(4) a <=b and b<=5

(5) not (a > b)

(6) a <=5 or b <=5

(7) a >10 or (a<10 and b>5)

(8) b<5 and not a>12

3. 把下面的命题用逻辑表达式表示。

(1) A 小于 B 或小于 C

(2) A 和 B 都大于 C

(3) A 和 B 中有一个小于 C

(4) A 是非正数

(5) A 是奇数

(6) B 不能被 A 整除

(7) 角 A 在第一或第三象限

(8) A 是一个带小数的正数，而 B 是一个带小数的负数

4. 由键盘输入三条线段的长度，求这三条线段所构成的三角形面积。

5. 编写出能计算下列各式的程序。x 值由键盘输入。

(1) $$y=\begin{cases}\sqrt{x}, & x\geqslant 0\\ \sqrt{-x}, & x<0\end{cases}$$

(2) $$y=\begin{cases}1, & x>0\\ 0, & x=0\\ -1, & x<0\end{cases}$$

(3) $$y=\begin{cases}3x^2+4x+5, & 0.5\leqslant x\leqslant 1.5\\ 1/(x-5), & 1.5<x<5\\ \log_2(-x), & x<0\\ \log_2(x-5), & x>5\end{cases}$$

6. 某校规定评定奖学金的条件是：

（1）学期考试 5 门功课，每门成绩都在 90 分或 90 分以上。

（2）学期考试 5 门功课总平均成绩为 90 分以上，其中有 1 门功课成绩为 100 分，1 门功课成绩不低于 75 分。

试编程输入一个学生的学号、姓名及 5 门功课成绩，输出该学生是否能评上奖学金的信息。

7. 按下述规定编写一个计税程序。

（1）销售额≤1000 元免税；

（2）1000 元<销售额≤2000 元，抽税 2%；

（3）2000 元<销售额≤5000 元，抽税 4%；

（4）销售额>5000 元，抽锐 4%外，超过部分再抽税 5%，且加收手续费 50 元。

8. 某航空公司规定：在 7～10 月如定票超过 20 张，优惠票价的 15%，20 张以下优惠 5%，在 1～5，11 月，定票超过 20 张，优惠 25%，20 张以下优惠 10%；其它月份按实际价格，设计一个程序，根据预定票张数和月份决定实际票价率。

9. 某旅社有三种等级的客房各 20 间，分别设在 3～4 楼(房号 301～310，401～410)，5～6 楼的客房每人每天收费 20 元，7～8 楼的客房每人每天收费 12 元。编程输入旅客姓名、住宿的房号及天数，计算出应付的住宿费。

10. 某商店进行商品优惠大展销，凡购货满 1000 元者优惠 8.5%，满 500 元者优惠 7%，满 200 元者优惠 5%，满 100 元者优惠 3%，100 元以下无优惠。编程输入顾客购买商品总金额，输出顾客实际应付的货款金额。

11. 输入一个数，编程输出该数是奇数还是偶数，如果是奇数则进一步判断它是否为 3、5、7 的倍数。

12. 某企业生产年产值为 1000 元，设年增长率为 3%，4%，5%，问 5 年后该企业的产值分别为多少？试编程进行计算。

13. 指出下列程序段中的错误。

(1)
```
DO WHILE A * A <A+A
    A=A+1
LOOP
```

(2)
```
FOR I=1 TO 5
    READ A,B,C
    FOR J=1 TO 3
        DATA 3 * PI
    NEXT J
NEXT I
```

(3)
```
FOR I=J+1 TO 2 * J STEP 0.1 * J
    PRINT I,J
    LET J=-J
NEXT I
```

14. 求下列程序的运行结果。

(1)
```
DO
    READ A,B
    IF A * B<=A+B THEN
        PRINT A,B
    END IF
LOOP UNTIL A=-1
DATA 5,-3,2,8,-6,10,2,-1,
  3,5,-1,-7
END
```

(2)
```
N=1
DO WHILE N<=10
    FOR K=1 TO N
        PRINT "$"
    NEXT K
    PRINT
    LET N=N+1
LOOP
END
```

15. 把下列式中□号内填入适合的数字，使等式相等。

(1) $\square3\times6528=3\square\times8258$

(2) $111111=111\times11\times\square1$

(3) □+□=□

□−□=□

□ * □=□

□/ □=□

－－－－－－－－－－－－－－－－

64

(4) $(\square*(\square3+\square))^2=8\square\square9$（第一个数不是 1）

16. 求下列数列的值。

(1) 1～100 之间奇数的对数及偶数的平方和

(2) 1+1/2+1/3+…+1/100 的和

(3) 1+5+9+13+17+…+101+105

(4) 1·2·3+2·3·4+3·4·5+…+109·110·111

(5) $1+2^1+2^2+2^3+2^4+\cdots+2^{63}$

17. 计算圆周率 π 的级数方法有以下几种，试编写出计算程序。

(1) $\pi/2=2/1\cdot2/3\cdot4/3\cdot4/5\cdot6/5\cdot6/7\cdot8/7\cdot8/9\cdots2N(2N-1)\cdot2N/(2N+1)\cdots$

(2) $\pi/4=1-1/3+1/5-1/7+\cdots+(-1)^{N-1}/(2N-1)+\cdots$

(3) $\pi/4=(1/2+1/3)-1/3(1/2^3+1/3^3)+1/5(1/2^5+1/3^5)-1/7(1/2^7+1/3^7)+\cdots$

(4) $\pi^2/6=1+1/2^2+1/3^2+1/4^2+\cdots+1/N^2+\cdots$

(5) $\pi^2/8=1+1/3^2+1/5^2+1/7^2+\cdots+1/(2N-1)^2+\cdots$

18. 一台天平配有 1，2，4，8，16，32，64，128，256，512 克等 10 种法码，现有重物在 1000 克之内(整数值)，在天平平衡的条件下，求出用了哪些法码？

19. 求出 1981～2000 年之间的素数年号。

20. 验证哥德巴赫猜想：每一个偶数都可以写

成两个素数之和，要求编一个程序对于从 4 到 500 为止的每一个偶数，都输出两个相应的素数。

21. 个位数是 6，且能被 3 整除的 5 位数有多少？

22. 有一个 4 位数字，前 2 位相同，后 2 位也相同，而这 4 位数正好是一个整数的平方，请编写一个程序打印出该数。

23. 试编写一个程序求出 11^{11} 的个位数、十位数、百位数各是多少？它们的和是多少？

24. 编写一个程序，将数 1,2,3,4,5 分成三个不同的数为一组，使该数当中的数字减去首数等于尾数，减去尾数等于首数。

25. 一个自然数以 6 结尾，如果把 6 移到第一位之前，所得的数是原数的 4 倍，求该数。

26. 假设我国工业产值增长率为每年 5%，7%，9%，11%，13%等，请计算出需要多少年工业产值可以增长 1 倍。

27. 李老师知道张老师年龄，而张老师不知道李老师年龄，故张老师问李老师的年龄，李老师说：“我年龄的平方与你的年龄之和等于 1053，而你的年龄的平方与我年龄之和等于 873，你计算一下”，请问李老师和张老师的年龄各是多少？

28. 小猴子摘来一些桃子，它吃了 1 半又吃了 1 只，第 2 天先吃了剩下的一半再多吃 1 只，第 3 天也如此，直到第 10 天就剩下 1 个桃子，问小猴一共摘了多少只桃子？

29. 用 100 元钱买 10 只鸡，母鸡 5 元 1 只，公鸡 3 元 1 只，小鸡 1 元 3 只，问应各买多少只？

30. 有一条长的阶梯，如果每步 2 阶，则最后剩 1 阶，如每步 3 阶则剩 2 阶，如每步 5 阶则剩 4 阶，每步 6 阶则剩 5 阶，只有每步 7 阶的最后才正好走完，一阶不剩，问这条阶梯最少共有多少阶？

31. 某试卷有 26 个问题，答对一题得 8 分，答错一题扣 5 分. 某个学生回答了全部 26 个问题，总分为 0 分，问他答对了多少题，答错了多少题？

32. 有 30 个人，其中有男人、女人、小孩。他们吃一餐午饭，男人每人化了 3 元，女人每人化了 2 元，小孩每人化了 1 元，共化了 50 元，问男人、女人、小孩各有多少？

33. 试编写程序，能分别显示输出如下式样的图样：

(1)

```
1×1=1
1×2=2  2×2=4
3×1=3  3×2=6  3×3=9
4×1=4  4×2=8  4×3=12  4×4=16
5×1=5  5×2=10  …  5×5=25
```

(2)

```
* * * * * * * * *
  * * * * * * *
    * * * * *
      * * *
        *
      * * *
    * * * * *
  * * * * * * *
* * * * * * * * *
```

(3)

```
5  5  5  5  5  5  5  5
4  4  4  4  4  4  4  4
3  3  3        3  3  3
2  2              2  2
1                    1
2  2              2  2
3  3  3        3  3  3
4  4  4  4  4  4  4  4
5  5  5  5  5  5  5  5
```

第5章　数组与数组运算

在科学计算和事务管理中，往往会遇到大批数据或信息需要处理，例如高考学生成绩的统计、分析和处理将涉及几万甚至几百万个数据。这些数据必须先存储后处理。按简单变量处理办法，每个数据用一个变量存储，变量总数显得庞大，甚至无法实现。为了解决批量数据的存储和处理，本章将介绍数组的有关概念、运算以及非数值处理的排序和检索等内容。

5.1　数组的概念

5.1.1　数组与下标变量

数据是程序处理和加工的对象。每个数据都有它的特性，且数据之间还存在着一定的关系。例如，某物理量测量获得：$\{P_1,P_2,P_3,\cdots,P_{n-1},P_n\}$；又如 n 次多项式：$a_nx^n+a_{n-1}x^{n-1}+\cdots+a_1x+a_0$ 的系数$\{a_n,a_{n-1},a_{n-2},\cdots,a_2,a_1,a_0\}$都是有关的数据序列。$P_1,P_2,\cdots,P_n$ 或 $a_n,a_{n-1},\cdots,a_1,a_0$ 都是数值。在多项式系数数列中，a_n 表示 x^n 项的系数，是数列中的第一个元素，a_{n-1} 是 x^{n-1} 的系数，是第二个元素，…，a_0 是 x^0 项的系数，是最后一个元素。在教学中，这样的数列称为向量。向量中的元素由一个标识符命名，用下标表达不同元素在数列中的位置，从而有效地解决了向量运算问题。计算机中如何解决相关有序数列的处理问题呢？True BASIC 语言将这类数表定义为数组。数组是由有限个同一基本类型数据构成，用一个标识符命名，称数组名，或数组变量。数组中每个数据称数组元素，并用下标表示元素在数组中的序位，即带下标的变量，称下标变量。如 A(I)，A＄(I)，称下标变量，A 是下标变量名，括号中的 I 表示下标。用 DIM A(7)，A＄(7)定义一个一维数组。A，A＄则为数组名。由于元素类型相同，每个元素占据存储大小一致，元素的下标直接与存储地址相关，程序用下标变量可直接访问数组中的特定元素，同时下标可以是数值表达式或变量，程序可用循环结构十分方便地进行整个数组的数据输入输出，或进行不同处理。可见，数组的引入不仅可解决批量数据存储命名的困难，更有意义的是将批数据组织，便于程序管理、操作处理。

数学中的一维向量用一维数组实现，元素数据存储在单下标变量中。

5.1.2　多维数组和多下标变量

在数学中经常遇到二维数列，如矩阵、线性方程组系数，都用行、列两个下标表示数据在数列中的位置。例如一个 3×3 的矩阵，

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

用 DIM A(3,3)二维数组定义，数组中元素用下标变量表示，如 A(I,J)(I=1,2,3，　J=1,2,3)表示 I 行、J 列元素。

二维平面数列也有以下特性：(1)元素数据的类型一致，即存储空间大小一致；(2)有限数据集合；(3)数据有序按行列顺序排列。在内存连续存区中元素排列采用行优先的原则。在处理数值或非工程设计中矩阵十分有用，True BASIC 提供了矩阵处理函数及运算规则。

同样有字符串为元素的二维数组。如某班同学座位安排、多门学科的考试成绩等。还会遇到三维、四维等多维数组，例如双层客车座位安排，除行、列编排外，还有层次，即用层次、行、列三个参数才能定位某一旅客座位。时空结合就有四、五维的情况。层、行、列、时间构成四维；日期、时间、层、行、列构成五维等。相应的元素用三下标变量、四下标变量或五下标变量表示。一般不希望定义过多维的数组，不仅因为多维数组不易形象表达，而且占用存储空间很大。True BASIC 允许最多定义 255 维的数组，且每维只一个元素。

例 5.1　某小组有 6 人，每人有学号和语文、数学两门成绩，要求计算每人的总分和平均分，并将每个人的总分和平均分数打印出来。程序如下：

```
DIM a(6,5)
FOR  i=1 TO 6
    FOR j=1 TO 3
      READ a(i,j)
    NEXT j
NEXT i
FOR  i=1 TO 6
    FOR j=2 TO 3
      LET a(i,4)=a(i,4)+a(i,j)
    NEXT j
    LET a(i,5)=a(i,4)/2
NEXT i
PRINT"No. ","Ch. ","Ma. ","tot. ","ave. "
FOR  i=1 TO 6
    FOR j=1 TO 5
      PRINT a(i,j),
    NEXT j
    PRINT
NEXT i
DATA 9201,85,97,9202,90,83,9203,77,65
DATA 9204,74,94,9205,96,82,9206,98,80
END
OK.
NO      Ch.     Ma.    tot.     ave.
  9201  85      97     182      91
  9202  90      83     173      86.5
  9203  77      65     142      71
  9204  74      94     168      84
  9205  96      82     176      88
  9206  98      80     178      89
OK
```

5.1.3　数组说明语句

数组是组织有序同类数据的构造型数据结构。为了让系统为它开辟并保留必要的连续存储空间，在使用之前必须先说明它的名称、类型、维数和容量等。

(1)数组说明语句 DIM

格式：DIM<数组名>[＄](<界说明>)[{,<数组名>[＄](<界说明>)}]

<界说明>为[<下界>TO/:]<上界>[{,[<下界>TO/:]<上界>}]

语句作用：用作定义数组，说明数组的类型、维数及元素的个数，以便系统留出足够的存储空间。

数组说明语句向系统提供如下信息：

① 数组名及数组类型。数组按元素的基本类型分为数值型和字符串型两类。例如：B(5,4)数组名为 B，类型为数值型数组。A＄(1:5)数组名为 A，类型为字符串数组。

② 数组维数。数组定义中，界说明的个数代表数组维数。例如，A(1 TO 5)为一维，B

(1 TO 3,2 TO 4)为二维,C(1 TO 2,5 TO 10,15 TO 30)为三维……。

③ 每一维的上下界,用来指定下标的取值范围。例如:一维数组 A(5),下标的下界是1,上界是 5;二维数组 C$(2 TO 3,10:15),第一维如上、下界分别为 2,3,第二维的下界为 10,上界为 15。

④ 数组的大小。数组元素多少由维数和每维元素数决定。例如:A(1 TO4,1:5)是 4 行,5 列的两维数组,共有 4×5=20 个元素。B(-10 TO 15,-8 TO 12),共有(15-(-10)+1)×(12+8)+1)=26×21=546。

数组大小(即元素数)的计算方法:对于一维元素数用(上界-下界+1)计算;对二维、三维等元素数,其总数是每维元素数的总乘积。

数组说明语句的几点说明:

① 数组名的命名规则与简单变量相同,但在同一程序块中不允许数组名与简单变量名相同。简单变量只代表一个数据,数组变量代表一个有关数据序列。

② 一个 DIM 语句,可以同时说明多个数组,数组之间需用逗号“,”隔开。

③ 界说明的个数,说明数组的维数。

多维数组的界说明之间用逗号分隔。由<界说明>说明数组每维元素的个数,即说明数组使用时下标取值的上、下界。

④ DIM 语句是一个非执行语句。在 DIM 语句中,界说明只能用数值,不允许使用变量或表达式。

⑤ 尽量避免定义空间太大的数组。例如:DIM A(1 TO 200,1 TO 200),这就要占用 39K 单元,对于内存较小的计算机已无法工作。

⑥ 数组定义的大小要适当,在某些情况下,当定义的数组无法事先确定其大小时,则可先定义数组名、维数和每维的下界。如 A(1,1),B(1)。在使用时,可以用 MAT 语句边输入边调整其上下界,改变数组的大小。

(2)数组的界说明

在 DIM 语句中定义数组某维下标的上、下界,意味着定义了在本程序中该数组在该维上可以使用下标值的范围。例如:DIM A(30),定义了 A(1)~A(30),共 30 个元素。

几点说明:

① True BASIC 允许根据需要定义下标的上、下界。如所含成绩的数组,可用学生的学号(9201)作为下界,而(9230)作为上界,即为 DIM A(9201 TO 9230),其结果仍是一个具有 30 个元素的数组,以便于从数组的下标中直接寻找到某学生的成绩。例如:想存储 1949~1992 年每年高校招生人数,并查询 1990 年招生人数。

```
DIM zhao(1949 TO 1992)
FOR i=1949 TO 1992
    INPUT zhao(i)
NEXT i
PRINT "zhao of 1990 is";zhao(1990)
```

显然这是很直观而方便的。

② 定义下标的上、下界的值应是-1E9 到+1E9 范围内的整数。

③ 不允许使用变量或表达式定义下标的上、下界。例如,DIM a(20 TO 45/5,15 TO 3×2),出错。

④ 允许用冒号代替 TO 分隔上界与下界。例如:DIM zhao(1949:1992)

⑤ 当下界为 1 时，可省略“〈下界〉TO/:”部分。例如：DIM(1:100,1:10)和 DIM a(100,10)等价。

(3)数组变量及下标变量

True BASIC 提供 MAT 语句，使用数组变量实现整个数组的输入、输出、传送和处理。应用中还经常需要使用下标变量随机访问数组中某个元素。

数组元素访问格式：<数组名>(<下标表达式>[{，<下标表达式>}])

其中，<下标表达式>是一个数值表达式，有自动取整功能的整数下标值。下标只能在数组定义式调整后的界范围内取值，否则超界出错。

例 5.2　将数字 1,2,3,4,5,6,7,8,9 先排成一个方阵，然后将此方阵转置。程序为：

```
DIM a(3,3)
FOR  i=1TO 3
    FOR j=1 TO 3
      READ a(i,j)
    NEXT j
NEXT i
FOR  i=1 TO 3
    FOR j=1 TO 3
      PRINT a(i,j);
    NEXT j
    PRINT
NEXT i
PRINT
FOR  i=1 TO 3
    FOR j=1 TO 3
      PRINT a(j,i)
    NEXT j
PRINT
NEXT i
DATA 1,2,3,4,5,6,7,8,9
END
```

……………………

```
OK. RUN
  1    2    3
  4    5    6
  7    8    9

  1    4    7
  2    5    8
  3    6    9
OK.
```

此例说明，数组元素的引用完全由其下标来确定。PRINT A(I,J)和 PRINT A(J,I)，虽只有一点差异，但打印的结果正好后面是前面的转置。

5.1.4　数组测试函数

True BASIC 提供了下列对数组大小进行测试的 6 个函数，这些测试函数都返回一个数值，这些数值可以用于任何数值表达式中。

Size(A)——数组 A 中当前的元素数。

Size(A,n)——数组 A 第 n 维的当前元素数。

Lbound(A)——一维数组 A 的下界。

Lbound(A,n)——数组 A 第 n 维的下界。

Ubound(A)——一维数组 A 的上界。

Ubound(A,n)——数组 A 第 n 维的上界。

上述函数均以数组变量作为参数，返回一个可量度的数值。

5.1.5　简单应用举例

例 5.3　一位数学家曾提出一道牛数递增的算题：有一头母牛，它每年年初生一头小母牛，每一头小母牛从第 4 年起，每年年初也生一头小母牛。问在第 50 年时，牛的头数共

有多少?

(1)找递推关系。依题意可列表如下:

年　数	1	2	3	4	5	6	7	8	9	…	50
牛的头数	2	3	4	6	9	13	19	28	41	…	?

设 n 表示年数($n\geqslant4$),U_n 表示牛的头数,由上表可得:

$$U_4=U_3+U_1=2+4=6$$

$$U_5=U_4+U_2=6+3=9$$

所以递推公式是:　$U_n=U_{n-1}+U_{n-3}\qquad(n\geqslant4)$

(2)设计思路。从递推关系出发,建立一个 50 个单元的一维数组 A(1:50)或 A(1 TO 50),将 2,3,4 先分别送入 A(1),A(2),A(3)中去,从 A(4)开始反复利用公式,即可求得 A(50)。程序如下:

```
DIM a(1:50)
LET a(1)=2
LET a(2)=3
LET a(3)=4
FOR i=4 TO 50
  LET a(i)=a(i-1)+a(i-3)
NEXT i
PRINT a(50)
END
.......................
OK. RUN
  262271568
```

例 5.4　10 个同学某门功课的考试成绩按分数高底排序输出,并求出平均成绩。

这是一个统计、排序问题,将 10 个学生的成绩从高分到低分按降序进行排列。

本题采用选择排序算法,用双重循环处理。外循环控制作 n－1 次扫视。每次按顺序更改扫视起点元素,在第一次扫视时,第一个数组元素 A(1)作为扫视起始,并将设为比较对象。假设第 I 次扫视,则 A(I)是扫视比较对象。内循环控制一次扫视的比较,以设定元素为比较对象,与位于其后的其余元素逐个比较。若被比较数小于 A(I)中的数,则取下一个元素比较:若被比较数大于 A(I)中的数,则两数交换,一直进行到内循环结束,一遍扫视完毕,A(I)保待了该次扫视后的最大值。

外循环再取下一个数组元素作比较对象,再进入内循环扫视一遍,剩余的元素依次进行,直到 N－1 次扫视完成,外循环结束,排序进行完毕。程序如下:

```
DIM name $ (10),A(10)
LET S=0
FOR I=1 TO 10
    READ name $ (I),A(I)
    LET S=S+A(I)
NEXT I
LET S=S/10
PRINT“I”,“name”,“A”
FOR I=1 TO 10
    FOR J=I+1 TO 10
        IF A(I)<A(J)  THEN
            LET T=A(I)
            LET A(I)=A(J)
            LET A(J)=T
            LET T$ =name$.(I)
            LET name$ (I)=name$ (J)
            LET name$ (J)=T$
        END IF
      NEXT J
    PRINT I,name$ (I),A(I)
NEXT I
PRINT
PRINT “S=”;S
DATA  “9201”,78,“9202”,90,“9203”,87
DATA  “9204”,76,“9205”,85
DATA  “9206”,64,“9207”,97,“9208”,70
DATA  "9209",80,"9210",62
END
............................
OK. RUN
I             name        A
  1           9207        97
  2           9202        90
  3           9203        87
  4           9205        85
  5           9209        80
  6           9201        78
  7           9204        76
  8           9208        70
  9           9206        64
 10           9210        62
S=78.9
OK.
```

5.2 数组输入输出

数组数据的传输可以通过以下两种方式进行：

(1)通过单重循环(对一维数组)或多重循环(对多维数组)，用 INPUT 语句从键盘上给每一个数组元素传送数据，或用 READ 语句把程序数据区中数据传送给数组元素，用 LET 语句把一个表达式的值传送(赋给)一个或几个数组元素，用 PRINT 语句把数组元素逐个输出。

(2)将数组作一整体进行数据传送。在 True BASIC 中提供了 MAT 语句，该语句起到上述作用。

5.2.1 数组输入语句

(1) MAT INPUT 语句

MAT INPUT 语句的格式：

MAT INPUT<数组表>

MAT INPUT PROMPT expr $:array1,array2…

MAT INPUT[PROMPT<提示串>:]<数组表>

其中，<提示串>是字符串表达式，作为待输入项的提示。<数组表>为数组变量，多个数组变量间用逗号分隔。

功能：该语句可以一次从键盘上读入某个完整的数组，并能读入多个数组。

例 5.5

```
DIM a(3,3)
MAT INPUT a
```

相当于执行以下程序段：

```
FOR I=1 TO 3
  FOR j=1 TO 3
    INPUT a(i,j)
  NEXT j
NEXT i
```

① 执行这个语句和 INPUT 语句相似，可以从键盘上将数组元素全部按“计程器顺序”一个一个赋值，而数组表是从左到右顺序赋值。执行时，首先显示一个“?”号，且在一个单独行上依次键入数据，并在每一个数据项之间用逗号隔开。各数组与数组之间用一个回车换行隔开。

② PROMPT<提示串>是一选择项，当它被选用时，提示串必须是一个字符串表达式，该语句执行时要求响应输入显示，代替通常显示的“?”。

③ 同 INPUT 语句一样，从键盘上打入的输入回答可以用一个逗号“,”结尾，以表示还有数据要输入。对于头尾有空格的字符串，或输入的字符串中具有逗号“,”和双引号“”或输入空字符串时，键入时必须用双引号括起来。

④ 当输入项少于数组元素，则显示“Too few input items,please add more. ”的信息，并显示？等待再键入。当输入项多于数组元素时，则显示“Too many input items. exceas ignored. ”信息，将多余项忽略。

例 5.6 演示 MAT INPUT 语句及响应输入。

```
DIM A(5),B(2,2)
MAT INPUT A,B
PRINT "thank you"
MAT PRINT A,B
END
......................
OK. RUN
? 1,2,3,4
```

```
Too few input items. please add more.
? 5
? 1,2,3,4,5
Too many input items. exceas igmored. thank you
  1   2   3   4   5
  1   2
  3   4
OK
```

(2)用 MAT INPUT 语句调整数组大小

数组使用前必须先用 DIM 语句定义数组的上下界。一旦定义，就确定了数组的大小。MAT INPUT 语句只要不改变数组的维数，可以改变其上下界，这样数组的大小根据实际的需要可以进行改变。改变的形式可以是常数，也可以是数值变量或表达式，但变量必须先赋值。界表达式必须包含在圆括号中，紧随数组变量。例如：

```
DIM A(5),B(3,5),C(3 TO 5,4 TO 8)
MAT INPUT A(2 TO 7)
MAT INPUT B(2 TO 8,5 TO 15)
LET I=2
MAT INPUT C(2 TO I+3,I TO I+10)
```

用 MAT INPUT 语句改变了数组 A 的上下界，但它的元素数未变。对于 B 数组和 C 数组，只是它们的维数未变，但上下界和元素的个数都改变了。在改变 C 数组时，使用了变量和表达式。

True BASIC 中的数组调整功能很有用处，因为 BASIC 语言中凡使用数组都必须先定义。例如解 N 阶线性方程 的程序，一开始就必须先定义一个 N×N 的二维数组 A(1 TO N,2 TO N)，以便存放方程系数。但这个二维数组定义多大呢？为了节省内存，总想定义得小一些，但将来又可能不够用户使用。定义得太大，用户用不了，造成浪费。为了使程序灵活地适合于用户的使用要求(解高阶或低阶线性方程组都通用)，先定义一个 A(1:1,1:1)即可，然而由用户自己根据需要用 MAT INPUT 语句改变其界和大小。

```
DIM A(1:1,1:1)
INPUT PROMPT"N:":N
MAT INPUT A(N,N)
⋮
```

对一维数组的上界可以更加灵活地进行调整。True BASIC 提供了一个 MAT INPUT A(?)语句，该语句是在数组的圆括号中用一个问号“?”代替，提示用户可以改变一维数组的上界(但不能改变其下界)。这条语句和以前讲述的 MAT INPUT A(2 TO 10)的不同之

处是：MAT INPUT A(2 TO 10)虽然可以随意改变数组 A 的上下界，当程序执行这条语句时，一定要输满 8 个数据，程序才往下执行；而用 MAT INPUT A(?)语句后，当程序执行到此句时，可以输入任意个数据，只要按一下回车键，对数组 A 的输入就算结束，接着执行下面的程序。

例 5.7 演示 MAT INPUT 调整数组。

```
DIM A(10),B(10)
MAT INPUT A(5)
MAT INPUT PROMPT"B(?):":B(?)
MAT PRINT A;
MAT PRINT B;
PRINT ubound(B);"items"
END
........................
OK. RUN
```

```
? 1,2,3,4
Too few input items. please add more.
? 5
B(?):5,6,7,8
  1  2  3  4  5
  5  6  7  8
  4 items
OK.
```

从此例可以看出，对数组 A 调整为 5 个元素，一定要输满 5 个数，而对数组 B 其定义为 10 个元素，然而 MAT INPUT B(?)作随意调整，输入 4 项，改变为 B(1:4)的数组。

注：当从键盘上响应输入若干数据后，用一个逗号结尾，此意表示还要继续输入数据。直到最后按回车键为止，行输入结束，数组输入才结束。

(3) MAT LINE INPUT 语句

格式：MAT LINE INPUT PRONPT〈提示串〉〈数组＄表〉

功能：将输入的一行字符送给字符串数组中的一个元素。字符行以回车符结束，一行字符串长度可达 32767 个字符。

说明：

① PROMPT〈提示串〉为选择项，其作用同 INPUT 语句一样，该提示串仅为第一个键入行显示，而每个后继输入则换行显示“?”提示符。

② 逐行输入给数组的每个元素，一行输入完后按回车键，出现“?”再输入下一行。

③ 所有数组必须是字符串数组。在 MAT LINE INPUT 语句中，多数组时各数组用逗号“,”隔开。

④ 可用 MAT INPUT 语句相同的方法调整数组，但 MAT LINE INPUT 不允许使用(?)随意调整方式。

例 5.8 把一些国家名从键盘上逐行输入，并按字母顺序排列好，且打印出来。国家名是：CHINA，KOREA，JAPAN，FRANCE，CANADA，ENGLAND，INDIA，AMERICA。

程序：采用改进选择排序算法(参看 5.3 节)

```
DIM name$(10)
set Zonewidth 8
INPUT PROMPT "number?":n
PRINT "Name them…?";
MAT LINE INPUT name$(n)
FOR I=1 TO N-1
  LET L=I
    FOR J=I+1 TO N
      IF Name$(J)<Name$(L) THEN LET L=J
    NEXT J
    LET a$=name$(I)
    LET name $(I)=name $(L)
    LET name$(L)=a$
  NEXT I
```

```
MAT PRINT name $
END
......................
OK. RUN
Number
Name them…? CHINA
? KOREA
? JAPAN
? FRANCE
? CANADA
? ENGLAND
? INDIA
? AMERICA
AMERIC CANADA CHINA ENGLAND FRANCE
INDIA JAPAN KOREA
OK.
```

(4) MAT READ 语句

格式:MAT READ <数组表>

功能:该语句是将数据表中的数据逐行读入一个数组或多个数组中。例如:

```
DIM A(2,3)
MAT READ A
DATA 1,2,3,4,5,6
```

相当于以下程序段:

```
DIM A(2,3)
FOR I=1 TO 2
    FOR J=1 TO 3
        READ A(I,J)
    NEXT J
NEXT I
DATA 1,2,3,4,5,6
```

说明:对于二维数组来说,列下标优先于行下标,即先增加第二个下标的值,全部完了以后再增加第一个下标的值,即以行为单位,先输入或输出行数小的行中的每一个元素,其顺序同一维数组。一行完了以后再进行下一行,直到数组全部满足为止。例如,数组A(2:5,3)输入输出的顺序为:

A(2,1)　A(2,2)　A(2,3)
A(3,1)　A(3,2)　A(3,3)
A(4,1)　A(4,2)　A(4,3)
A(5,1)　A(5,2)　A(5,3)

对于多维数组,最后一个下标变化最快,其次是它前面的一个下标,变化最慢的是第一个下标。

在 MAT READ 语句中若有多个数组,则它们之间用逗号隔开。最左边数组读入完成后,再读入下一个数组,最右边数组最后读入。

MAT READ 语句也可用 MAT INPUT 语句相同的方法调整数组上下界,但没有 A(?)功能,例如:

```
DIM A(1,1),B(1,1)
MAT READ A(3,5),B(6 TO 10,3 TO 6)
```

A,B 数组在定义时都只定义了一个元素,但在读入时,分别读入 3×5 和 5×4 个数据,既改变了数组的上下界,也改变了数组的大小。

5.2.2 数组输出语句

(1) MAT PRINT 语句

一般形式:MAT PRINT<数组>[{,/;[<数组>]}]

其中，<数组>为数组变量。多个数组用逗号或分号分隔。

功能：执行 MAT PRINT 语句可显示出各数组的所有元素。

说明：

① MAT PRINT 语句是将数组中的元素逐行输出的，每输出一行后自动换行。

② MAT PRINT 语句中的逗号“，”或分号“；”是指定其前面数组中的每个元素的域格式或紧凑格式输出。只输出一个数组元素时，如数组变量后不带“，”或“；”时，数组元素按域格式输出。

③ MAT PRINT 无运算功能。

例 5.9　数组变量输出格式演示。

```
DIM A(2,4),B(4,2)
FOR I=1 TO 2
  FOR J=1 TO 4
    READ A(I,J)
    LET B(J,I)=A(I,J)
  NEXT J
NEXT I
MAT PRINT A,
MAT PRINT A;B,
MAT PRINT A
DATA 1,2,3,4,5,6,7,8
END
……………………
OK. RUN
```

```
1       2       3       4
5       6       7       8

1     2     3     4
5     6     7     8

1       5
2       6
3       7
4       8

1       2       3       4
5       6       7       8
OK.
```

例 5.10　数组输出演示。

```
DIM A(2,3,4)
FOR I=1 TO 2
FOR J=1 TO 3
  FOR K=1 TO 4
    LET A(I,J,K)=100 * I+10 * J+K
    NEXT K
  NEXT J
NEXT I
MAT PRINT A;
END
```

```
……………………
OK. RUN
  111  112  113  114
  121  122  123  124
  131  132  133  134
  211  212  213  214
  221  222  223  224
  231  232  233  234
OK.
```

MAT 语句输出三维数组时，是按“逐页逐行”原则输出的，每输出一页，空一行。对 M 维数组，MAT PRINT 语句输出其元素的次序是先变化第 M 维下标，再依次变化第 M－1 维，M－2 维，直到第一维的下标。一维数组则可以看成只有一行的二维数组。此原则也适用于 MAT INPUT 和 MAT READ 语句。

(2) MAT PRINT USING 语句

格式：MAT PRINT USING <格式串> format $ ：<数组表>array1，array2，…

功能：MAT PRINT USING 语句的各数组及数组的各元素输出次序同 MAT PRINT 语

句，而输出格式则按〈格式串〉指定的格式输出。

如果元素个数超过格式串范围，格式串将被再使用并从起始处开始。

使用 MAT PRINT USING 时，数组中每一行在一个新的行上开始，如果显示几个数组，在数组间必须留下一个空行。

例 5.11　格式化数组输出演示。

```
DIM A(3,4)
FOR I=1 TO 3
  FOR J=1 TO 4
    LET A(I,J)=10*I+J
  NEXT J
NEXT I
MAT PRINT USING"# # #. # $ # #"; A
```

```
END
........................
OK. RUN
  11.0   $12   13.0   $14
  21.0   $22   23.0   $24
  31.0   $32   33.0   $34
OK.
```

5.2.3　数组变量赋值，MAT 语句与运算功能

(1) 用表达式赋值数组

格式：MAT <数组>=<表达式>

功能：将常量或已赋过值的变量赋给数组的每一个元素。赋值号右侧如果是表达式，则表达式必须在圆括号中，否则出错。若给两个数组赋同一值，必须用两条赋值语句分别赋值。

例 5.12　用表达式给数组赋值。

```
DIM a(2,3),b(2,4),c$(A)
MAT a=10
MAT b=(sqr(25)*2/3+1)
MAT c$=("no"&"thing"&"   ")
MAT PRINT a;b;c$;
END
OK. RUN
```

```
10   10   10
10   10   10

4.33333   4.33333   4.33333   4.33333
4.33333   4.33333   4.33333   4.33333

nothing   nothing    nothing   nothing
```

(2)数组赋值及调整

MAT<数组 1>=<数组 2>

若数组 1 和数组 2 维数相同且每维大小对应相等(上下界不一定相同)，则数组 2 将元素内容一一对应传送给数组 1；当数组 1 和数组 2 维数相同但各维数大小不同，则按数组 2 自动调整数组 1 的上界，使之与数组 1 大小相同。

例 5.13　数组赋值及调整。

```
DIM A(3,3),B(1,1),C(10 TO 11,20 TO 21)
FOR I=1 TO 3
    FOR J=1 TO 3
      LET A(I,J)=10*I+J
    NEXT J
NEXT I
MAT B=A
MAT C=A
```

```
PRINT "ARRAY A:"
MAT PRINT A;
PRINT "(B=A),ARRAY B:"
MAT PRINT B;
PRINT "(C=A)ARRAY C:"
FOR I=10 TO UBOUND (C,1)
  FOR J=20 TO UBOUND (C,2)
  PRINT "C(";I;",";J;")=";C(I,J);
```

```
NEXT J
PRINT
NEXT I
END
......................
OK. RUN
ARRAY A:
  11  12  13
  21  22  23
  31  32  33
```

```
(B=A)ARRAY B:
  11  12  13
  21  22  23
  31  32  33
(C=A)ARRAY C:
C(10,20)=11 C(10,21)=12 C(10,22)=13
C(11,20)=21 C(11,21)=22 C(11,22)=23
C(12,20)=31 C(12,21)=32 C(12,22)=33
OK.
```

由运算结果可以看出：执行 MAT C=A 后，A 改变了 C 的上界，没有改变 C 的下界。

(3) 矩阵运算、矩阵函数和内部数组

① 矩阵加、减运算。数学中的矩阵和向量都可用数组表示。矩阵是一个二维元素为数值的数组；向量是一个一维的数值数组，亦可看作是单行（或单列）的矩阵。

在 True BASIC 中，可以用 MAT 语句作矩阵的加、减运算，运算遵循数学上的矩阵运算法则，要求参加运算的矩阵行、列数相同，位于相同行、列上的对应元素作加、减。例如 A,B,C 表示三个具有相同维数及大小的数组，用下述语句表示矩阵加、减运算：

```
MAT A=B+C
MAT A=B-C
MAT A=A+B
```

如果被赋值数组维数不满足矩阵运算的维界要求，则自动被调整，但仅上界被调整，下界不变。两个矩阵相加、减是将其对应元素相加、减。

例 5.14　矩阵加、减运算。

```
DIM A(2,3),B(2,3),C(2,3),D(2,3)
MAT A=20
MAT B=10
MAT C=A+B
MAT D=A-B
MAT PRINT C;D;
END
......................
```

```
OK. RUN
  30  30  30
  30  30  30

  10  10  10
  10  10  10
OK.
```

② 数值乘矩阵的运算

格式：MAT<矩阵 A>=<表达式>*<矩阵 B>

其中，<表达式>可为常数或已赋值的变量，或包括在圆括号中的数值表达式。其功能是将矩阵 B 的每一个元素都乘以表达式的计算值，然后赋值给矩阵 A，如 MAT A=3*B。

例 5.15　矩阵数积运算。

```
DIM A(2,3),B(2,3),C(2,3)
MAT A=10
MAT B=3*A
MAT C=(sqrx(25)*4)*A
MAT PRINT A;B;C;
```

```
END
......................
OK. RUN
  10  10  10
  10  10  10
```

```
                                   200  200  200
  30   30   30                     200  200  200
  30   30   30                  OK.
```

③ 矩阵相乘运算

格式：MAT<矩阵 A>=<矩阵 B>*<矩阵 C>

矩阵相乘应遵循数学矩阵相乘的规则，即如果 A 矩阵是一个 L×M 矩阵，B 是一个 M×N 矩阵，A×B 所得的结果则是一个 L×N 矩阵。如果 A 是一个含 M 个元素的向量（1×M 的数组），B 是一个 M×N 的矩阵，则结果是一个 N 元素的行向量。同矩阵加、减运算一样，它可隐蔽地调整目标数组。

例 5.16　矩阵相乘运算。矩阵：

$$A=\begin{pmatrix}1 & 2\\3 & 4\\5 & 6\\7 & 8\end{pmatrix},B=\begin{bmatrix}1 & 2 & 3\\4 & 5 & 6\end{bmatrix},$$

求 A×B。程序如下：

```
DIM A(4,2),B(2,3),C(4,3)                ........................
MAT READ A,B                            OK. RUN
MAT C=A*B                                 9   12   15
MAT PRINT C;                             19   26   33
DATA 1,2,3,4,5,6,7,8                     29   40   51
DATA 1,2,3,4,5,6                         39   54   69
END                                     OK.
```

④ 内部常数数组和矩阵函数。在 True BASIC 中提供一些常数数组，这些数组值可用在 MAT 赋值语句的赋值号右边。如果使用一个具有明确界限（带有参数）的数值组，则目标矩阵的形状、大小能按照即将赋予的数值组值的大小和形状重新调整。如果使用一个不带参数的内部常数数组，则目标数组保持自己的形状和大小。如果给出下界参数，则它必须与目标数组的下界匹配，否则出错。

(a) CON——给目标数组所有元素全赋 1。

(b) CON(x,y,…)——给目标数组所有元素全赋 1，并调整为 x×y×…大小的数组。

(c) IDN——给目标数组赋给单位阵（主对角线上元素均为 1，其它元素为 0 的方阵）。要求目标数组是行列数相等的二维数组。

(d) IDN(x)——给目标数组赋单位阵，这是一个 x×x 的方阵，要求目标数组是二维的。

(e) NUL $——给目标数组所有元素赋空字符串。

(f) NUL $(x,y,…)——给目标数组所有元素赋空字符串，调整为 x×y×…数组。

(g) ZER——给目标数组所有元素全赋 0。

(h) ZER(x,y,…)——给目标数组所有元素全赋 0，调整为 x×y× 的数组。

以上不带自变量的函数由被赋值数组决定大小，而带自变量的函数由函数自变量决定被赋值数组的大小，要求被赋值数组的维数与函数自变量个数相等。

例 5.17　用常数数组赋值。

```
DIM A(2,3),B(2,2),C(8,8),D(3,6)
MAT A=CON(4,3)
MAT B=CON
LET N=5
MAT C=IDN(N)
MAT D=ZER(2,4)
MAT PRINT A;B;C;D;
END
……………………
OK.RUN
  1  1  1
  1  1  1
```

```
1  1  1
1  1  1

1  1
1  1

1  0  0  0  0
0  1  0  0  0
0  0  1  0  0
0  0  0  1  0
0  0  0  0  1
OK.
```

⑤ 矩阵函数

True BASIC 提供两个矩阵函数：INV、TRN。函数返回一个矩阵，可出现在矩阵赋值号的右侧。

(a) INV(A)——返回正方矩阵 A 的逆矩阵。如果 A 是一个奇异矩阵或近似于一个奇异矩阵，则返回一个出错信息。

例 5.18　求下列三元一次方程组的解：

$$\begin{cases} 3X_1 + 2X_2 + X_3 = 14 \\ X_1 + X_2 + X_3 = 10 \\ 2X_1 + 3X_2 - X_3 = 1 \end{cases}$$

分析：求线性方程组的解可转换为矩阵的运算，其形式为：

$$C=A*X$$

$$\begin{bmatrix} 14 \\ 10 \\ 1 \end{bmatrix} = \begin{bmatrix} 3 & 2 & 1 \\ 1 & 1 & 1 \\ 2 & 3 & -1 \end{bmatrix} * \begin{bmatrix} X_1 \\ X_2 \\ X_3 \end{bmatrix}$$

若在等式 C=A∗X 两边乘以 A^{-1}(A^{-1} 是 A 矩阵的逆矩阵)，得：

$$A^{-1}*C=A^{-1}*A*X$$

而矩阵 A 与其逆矩阵 A^{-1} 相乘得单位矩阵，即 $A^{-1}*A=I$(I 表示单位矩阵)。所以，

$$A^{-1}*C=I*X \qquad 即\ X=A^{-1}*C$$

利用 INV(A)函数可求出矩阵的逆矩阵，若用 MAT,B=INV(A)。于是，X=B∗C。这样，把线性方程组的解转换为两矩阵的乘积。程序如下：

```
DIM A(3,3),B(3,3),C(3,1),X(3,1)
MAT READ C,A
DATA 14,10,1
DATA 3,2,1,1,1,1,2,3,-1
MAT B=IND(A)
MAT X=B*C
PRINT "MAT C"
MAT PRINT C;
PRINT "MAT A"
MAT PRINT A;
PRINT "INVERSE OF MAT A"
MAT PRINT B;
PRINT "MAT X"
MAT PRINT X;
```

```
END
……………………
OK.RUN
MAT C
  14
  10
  1
MAT A
  3  2  1
  1  1  1
  2  3  -1
INVERSE OF MAT A
 .8   -1.   -.2
-.6    1.    .4
-.2    1.   -.2
MAT X
 1
 2
 7
OK.
```

(b) TRN(A)——返回矩阵 A 的转置矩阵

若矩阵 $A_{m\times n}$ 与 $B_{n\times m}$ 存在 $a_{ij}=b_{ji}$，则 A，B 互为转置。例如：

$$A=\begin{bmatrix}1&2\\3&4\\5&6\end{bmatrix},B=\begin{bmatrix}1&3&5\\2&4&6\end{bmatrix},$$

则 A，B 互为逆矩阵。

例 5.19　矩阵转置

```
DIM A(2,3),B(2,3)
MAT READ A
DATA 1,2,3,4,5,6
PRINT"MAT A"
MAT PRINT A;
MAT B=TRN(A)
PRINT"MAT B"
MAT PRINT B;
END
……………………
```

```
OK.RUN
MAT A
 1  2  3
 4  5  6
MAT B
 1  4
 2  5
 3  6
OK.
```

⑥ 数组函数 DET 和 DOT。DET 和 DOT 返回一个数值。DET(A)用以返回方阵 A 的行列式的值。如果 A 是奇异矩阵，则 DET(A)=0；当 DET 函数不带参数，则返回上一次用 INT(A)函数求逆矩阵的行列式 A 的值；如果未对任何矩阵求逆，则 DET 返回 0。

例 5.20　求下列方程组的解：

$$\begin{cases}x+2y-z=1\\2x-3y+z=0\\4x+y-z=3\end{cases}$$

```
DIM A(3,3),B(3,3),C(3,1),x(3,1)
MAT READ A,C
DATA 1,2,-1,2,-3,1,4,1,-1
DATA 1,0,3
IF DET(A)<>0 THEN
    MAT B=INV(A)
    MAT x=B*C
    FOR i=1 TO 3
        PRINT "x";i;"=;"x(i,1)
    NEXT i
    PRINT
ELSE
    PRINT "|A|=";DET(A);"ERROR!"
END IF
END
……………………
```

```
OK. RUN
|A|=0 ERROR
OK.
```

函数 DOT(A,B)——返回向量(一维数组)A 和 B 的点乘积,向量 A 和 B 必须具有相同的元素数,其结果为一个单个的数。

例 5.21　求向量 A=(1,2,3,4)和 B=(1,2,3,4)的点积。

```
DIM A(4),B(4)
MAT READ A,B
DATA 1,2,3,4,1,2,3,4
LET D=DOT(A,B)
MAT PRINT A;B;
PRINT D
END
........................
OK. RUN
  1  2  3  4  1  2  3  4
  30
OK.
```

5.3　排序与检索

5.3.1　数据排序

排序与检索是数据处理中最频繁的两项应用。按照预先规定的准则,如升序或降序等,把数据有次序地排列起来的过程称为"排序"。排序方法有以下几种。

(1) 选择排序

选择排序的基本思想是:对所有的数据作第一次扫视,首先拿出数组中的第一个元素 A(1)依次与 A(2),A(3),…,A(N)比较。每比较一次,取大的(假设降序排列)数据送 A(1);当第一轮扫视结束,A(1)中就存放 N 个数中最大的一个数,可见第一次扫视对 N 个数据的数组要作 N－1 次比较。然后进行第二次扫视,将第二个数据 A(2)取出,依次与 A(3),A(4),…比较,每比较一次,取大的数据放入 A(2);当第二轮比较结束,A(2)中就存放着 N 个数中次大的数,第二次扫视则要比较 N－2 次。依次类推,共扫视 N－1 次,完成排序工作。

评价一个排序算法的优劣,通常是以其排序速度、占用空间大小等因素衡量的,重点是速度。排序中花费时间多少取决于两种运算:①比较次数多少的运算,②交换次数多少的运算。就选择排序法而言,对于 N 个数据,在排序时需要 N(N－1)/2 次比较。比较后,可能要进行交换,最理想状态为 0 次(也就是说本身已按规定要求排好,无需再排序)。最多为 N－1 次,平均次数为(N－1)/2 次。因此,在一次排序过程中需要比较、交换次数为[N(N－1)/2][(N－1)/2]=N(N－1)(N－1)/4 次。

为了减少交换次数,可以对上述程序稍作修改。例如,设置一个指针 K,每轮中仅记下最大一个元素的下标值,仅交换一次,这样排序中平均最多交换 N－1 次,最少交换 0 次,因而提高了排序效率。

(2) 冒泡法排序(相邻比较交换法)

N 个数据按升序排列的冒泡排序算法基本思想是:依照从小到大的排序原则,先比较 A(1)和 A(2),若同序〔即 A(1)<=A(2)〕就不动;若逆序〔即 A(1)>A(2)〕就交换 A(1)与 A(2);接着比较 A(2)(新的)与 A(3),与以前一样,同序位置不变,逆序就交换……,依次进行,直到将 A(N－1)与 A(N)比较并交换完,完成了第一轮扫视。然后,按同样规则再从头进行第二轮扫视,直到进行完第 N－1 轮扫视为止,排序完成。

在冒泡排序过程中，象烧开水时壶中水的汽泡一样，小的汽泡每扫视一轮向上游一位，而大的汽泡往往下沉，而且每经一轮扫视最大的元素沉淀到底部，具体过程如图 5.1 所示。

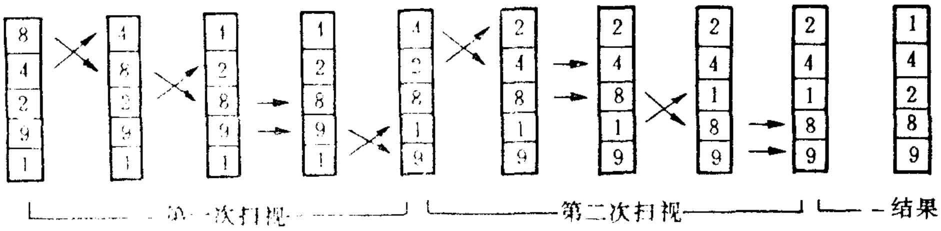


图 5.1

例 5.22　将 10 个人的数学考试成绩从高到低排序(采用冒泡算法)。程序为：

```
DIM S(10)
LET N=10
FOR I=1 TO N
READ S(I)
NEXT I
FOR I=1 TO N
  FOR J=1 TO N-I
    IF S(J)>S(J+1) THEN
      LET T=S(J)
      LET S(J)=S(J+1)
      LET S(J+1)=T
    END IF
  NEXT J
  PRINT I;S(I),
NEXT I
PRINT
DATA 90,85,74,93,66,100
DATA 87,78,96,72
END
```

交换法排序需要比较、交换的次数分析：如果排序前数列已按正序排好，则经一轮，即 N—1 次比较，交换次数为 0，便可完成排序。最坏的情况是初始序列为逆序，比较次数为 N(N—1)/2 次，而且每比较一次都要进行一次交换，故交换次数也最大，为 3N(N—1)/2。

(3) 插入排序

插入排序是按数据的升降规则将未排序数据依次取出，并逐一插入已排序的适当位置，直到将全部数据都插入为止的排序方法。因插入数据的方式不同，形成了以下两种不同的插入排序方法：

① 直接插入排序。设数组 A(N)中已放好 N 个无序数据，将其从小到大排序的方法是：首先将数组 A(N)中的 A(1)作为一个已排序好的集合〔当然 A(1)就其本身的排序既是最大，也是最小，没有其它元素作比较〕。然后取出 A(2)和 A(1)这一集合元素作比较，如果 A(2)>A(1)，则二者位置不变；如果 A(2)<A(1)，则交换 A(1)和 A(2)的位置，形成一个按规则的排序集合{A(1)，A(2)}。接着，取出数组中的 A(3)，与集合进行比较，只需用穷举法对已排序子序列按逆序逐一测试便可，将(3)插入适当位置。到此，{A(1)，A(2)，A(3)}已是按升排序的集合。再将 A(4)…依次插入排序集合中。在插入过程中如何解决元素移动是个重要问题。为了保证数据不丢失，首先将待插入的元素装入一个辅助单元 T 中，利用中间变量实现边比较边移动，以填充空位。

例 5.23　用直接插入算法将 7 个数据按升序排列。程序为：

```
DIM A(1)
INPUT PROMPT"Enter numbers of element: "N
MAT READ A(N)
FOR I=2 TO N
  LET T=A(I)
LET J=I-1
DO UNTIL J<1 OR A(J)<T
  LET A(J+1)=A(J)
  LET J=J—1
LOOP
  LET A(J+1)=T
NEXT I
```

```
MAT PRINT A;
DATA 13,9,11,15,7,3,5
END
........................
```

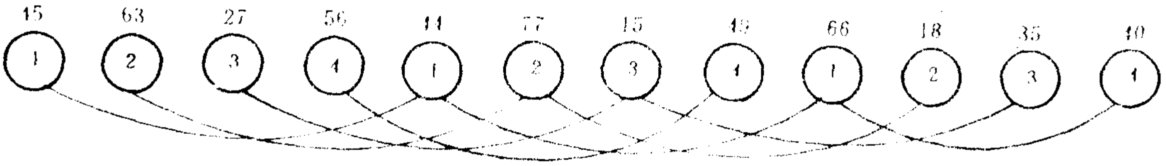
```
OK. RUN
Enter numbers of element:7
3  5  7  9  11  13  15
OK.
```

插入排序的最浅显理解是玩扑克牌对抓牌按顺序一一插入。直接插入排序的算法简洁，容易实现。完成一次排序所需的最少比较次数为 N－1 次，最多的比较次数为(N－1)(N＋2)/2 次；数据在排序中可能的移动次数最少为 2(N－1)次，最多移动 $N^2/2$ 次。

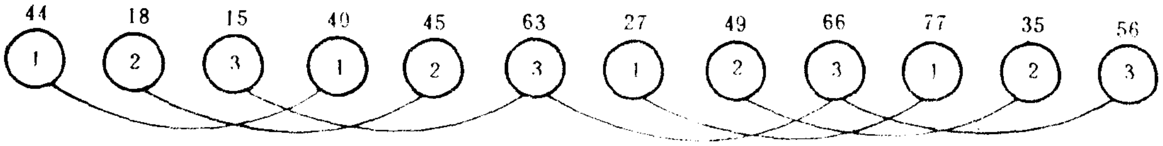
② 对分插入排序。在插入排序中经常用到对分插入排序法，这种方法是将要插入的数据按对分寻找法在排序元素集合中查找插入位置，这样可把总的比较次数减少到 $N\log_2 n$ 次。有关对分检索算法参看下节。

(4) 希尔(shell)排序

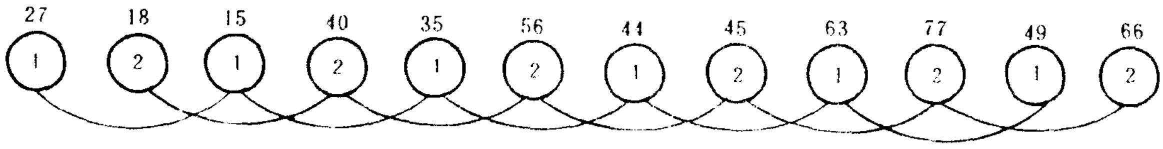
基本思想是：把一个较大的未排序的数据序列分割为若干较小的子序列，分别进行直接插入排序，并逐步扩大整个数列的有序性，直到完全有序。具体做法是：把整个数列按照某一个增量值(假设为 X)选取元素，划分成 X 个小的子序列，在各子序列内进行排序。第一轮排好后，将新排的数列重新按 X－1 增量分成 X－1 个新子序列，每个子序列又进行各自的排序，依次做下去，直到最后的增量为 1。例如，现有 12 个数：45,63,27,56,44,77,15,49,66,18,35,40；初始数列为：


45
63
27
56
44
77
15
49
66
18
35
40
1
2
3
4
1
2
3
4
1
2
3
4

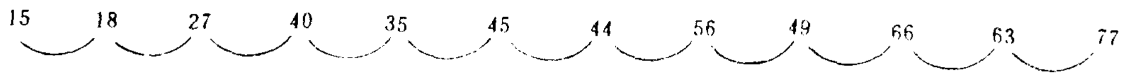
第一次增量为 4，将原序列分 4 组。其中：45,44,66 为第 1 组；63,77,18 为第 2 组；27,15,35 为第 3 组；56,49,40 为第 4 组。各组分别插入排序为 44,45,66 为第 1 组，18,63,77 为第 2 组；15,27,35 为第 3 组；40,49,56 为第 4 组。数列为：


44
18
15
40
45
63
27
49
66
77
35
56
1
2
3
1
2
3
1
2
3
1
2
3

第二次增量为 3，将新数列分成 3 组。其中：44,40,27,77 为第 1 组；18,45,49,35 为第 2 组；15,63,66,56 为第 3 组。各组分别插入排序为：27,40,44,77 为第 1 组；18,35,45,49 为第 2 组；15,56,63,66 为第 3 组。新序列为：


27
18
15
40
35
56
44
45
63
77
49
66
1
2
1
2
1
2
1
2
1
2
1
2

第三次增量为：将新数列分成 2 组。其中：27,15,35,44,63,49 为第 1 组；18,40,56,45,77,66 为第 2 组。各组分别插入排序为：15,27,35,44,49,63 为第 1 组；18,40,45,56,66,77 为第 2 组。新序列为：


15
18
27
40
35
45
44
56
49
66
63
77

第四次增量为 1，进行插入排序，最终数列为：15,18,27,35,40,44,45,49,56,63,66,77。

由上述可以看出，当增量为 X 时，具体分成子序列的方法是：

A(1)　A(X+1)　A(2X+1)…

A(2)　A(X+2)　A(2X+2)…

⋮

A(X)　A(2X)　A(3X)…

各组进行插入排序……

例 5.24　希尔排序程序。

```
DIM A(12)
FOR I=1 TO 12
  READ A(I)
NEXT I
LET N=UBOUND(A)
LET X=4
DO WHILE X>=1
  FOR R=1 TO X
    FOR I=X+R TO N STEP X
      LET Y=A(I)
      LET J=I-X
      DO WHILE J>0 AND Y<A(J)
        LET A(J+I)=A(J)
        LET J=J-X
      LOOP
      LET A(J+X)=X
      NEXT I
    NEXT R
    LET X=X-1
LOOP
MAT PRINT A
DATA 45,63,27,56,44,77
DATA 15,49,66,18,35,40
OK.RUN
  15   18   27   35   40   44   45   49   56
  63   66   67
OK.
```

希分排序法的排序速度较快，其平均比较次数和平均移动次数为 $N^{1.3}$ 倍左右。

(5) 排序关键字

在排序过程中，经常碰到排序关键字问题。例如，有一批学生记录，内容包括学号、期中成绩、期末成绩、总分等项。现将这批记录排序。当然，我们首先要问按什么标准排序，如果按学号排序，那么学号就称为排序关键字，假设学号不会有重复，那么，这种排序只有唯一的排序方式。如果按总分排序，则总分就称排序关键字。那么几个人的总分相等又该怎么办？这就要确定次排序关键字。这时，我们可以将学号作为次排序关键字。这样就出现了双关键字排序，其中一个主关键字(总分)，一个次关键字(学号)。如果着眼点放在分数上，次关键字可以不选学号，而选期末成绩。若总分和期末成绩都相等又该怎以办？可以再选学号作次关键字，这样就出现了三个关键字排序，其中一个主关键字，两个次关键字，且次关键字中还分第一次关键字(期末成绩)和第二次关键字(学号)，这就称为多关键字排序。

总分和期末成绩要求从高分到低分排序，即按升序排序，而学号要求从小到大排序，即按升序排序，这就是所谓关键字的升降。

排序种类很多，在此不一一列举，读者可参阅有关资料。

5.3.2　数据检索

检索是在数据集合中查找需要的数据。检索方法有：

(1)线性查找法。将数据集合依次存放在数组中，然后根据待查数据从数组的第一个元素开始逐个比较，若查到相同的，则查找成功并输出其序号，不然继续查找。如果到最后一个元素都未找到，则查找不成功，输出一个标志。

例 5.25　顺序检索算法程序。

```
DIM A(50)
INPUT PROMPT"Enter numbers of element:";N
MAT READ A(N)
MAT PRINT A(N);
INPUT V
FOR I=1 TO 50
  IF V=A(I) THEN
    PRINT A(I)
  END IF
NEXT I
PRINT"NOT FOUND!"
DATA A1,A2,…A50
END
```

线性查找实际上是顺序查找。可以看出，查找次数最少为 1，最多为 N，平均为 N/2。这种查找方法大多已不采用。

（2）折半查找法，又称对分查找法。算法的思想是：先把数据按升序（或降序）排序；将排后的数据序列的第一个数据位置定为查找下界，最后一个数据位置定为查找上界，对分找到中位数据，并将数据分成两部分，第一部分是下界～（中位－1）的数据，如按升序排列时，这部分数据皆小于中位数据，第二部分是（中位＋1）～上界数据，这部分皆大于中位数据；将中位数据与待查数据比较，若相等则查找成功，如果中位数据＞待查数据，则抛弃第二部分数据，将第一部分的下界定为查找下界，再按对等法重复上述过程，继续查找；如果中位数据＜待查数据，则抛弃第一部分，修改查寻下界，将第二部分再重复对分查找过程。查找可能有两种结果：查找成功，找到该数据及其在序列中的序号；查找失败，数据序列中不存在该数据。算法中当出现下界＞上界时，标志查找失败。对分查找法对半剔除，收敛快，效率高；由于该方法按数据排序位置对分确定中位数据，下标与元素有唯一对应关系，用数组实现对分查找算法，中位应该是整数的下标值。

例 5.26　已知一组按学号排序的学生记录（学号不连续），内容包括学号、性别、年龄、宿舍号码，使用对分法查找某一学生的全部记录。程序如下：

```
DIM NUMB(300),SEX$(300),AGE(300)
DIM ADD$(300)
LET N=0
DO WHILE MORE DATA
    LET N=N+1
    READ NUMB(N),SEX$(N),AGE(N)
READ ADD$(N)
LOOP
INPUT PROMPT "Enter NO.";X
LET F=0
LET L=LBOUND(NUMB)
LET H=N
DO UNTIL F=1 OR H<L
    LET M=INT((L+H)/2)
    IF NUMB(M)=X THEN
        LET F=1
        ELSE IF NUMB(M)>X THEN
          LET H=M-1
        ELSE
          L=M+1
        END IF
    LOOP
IF F <> 1 THEN
  PRINT "NOT FOUND"
ELSE
    PRINT NUMB(M),SEX$(M),AGE(M),
    PRINT ADD$(M)
END IF
DATA 93001,MALE,20,NO.102,92004
DATA FEMALE,19,NO.104…
END
```

5.4　数组应用

例 5.27　某商店现有 10 种商品，其库存见表 5.1，要求按进货价和销售价统计出商品积压资金。

表 5.1

编号和单价	商品编号									
	1	2	3	4	5	6	7	8	9	10
库存量	50	150	300	90	75	65	80	180	250	370
进货单价	4.50	3.00	10.00	50.00	90.00	5.00	83.00	15.00	35.00	56.00
销售单价	5.80	4.50	14.00	56.00	100.00	7.50	95.00	21.00	45.00	65.00

按进货价统计商店积压资金为：$\sum_{1}^{10}$库存量×进货单价；按销售价统计商品积压资金为：$\sum_{1}^{10}$库存量×销售单价。设置 3 个一维数组：A(10)，B(10)，C(10)，分别存放库存量、进货单价，销售单价。

程序如下：

```
DIM A(10),B(10),C(10)
LET N,ZJ1,ZJ2=0
MAT READ A,B,C
FOR I=1 TO UBOUND(A)
    LET ZJ1=ZJ1+A(I) * B(I)
    LET ZJ2=ZJ2+A(I) * C(I)
NEXT I
PRINT "ZJ1=";ZJ1,"ZJ2=";ZJ2
DATA 50,4.50,5.80,150,3.00,4.50
DATA 300,10.00,14.00
DATA 90,50.00,56.00,75,90.00
DATA 100.00,65,5.00,7.50
DATA 80,83.00,95.00,180,15.00
DATA 21.00,250,35.00,45.00
DATA 370,56.00,65.00
END
```

本程序也可以定义一个二维数组，使用矩阵函数，请读者自行考虑。

例 5.28　拉格朗日插值。设有离散点(X_0,Y_0)，(X_1,Y_1)，…，(X_N,Y_N)，这 N+1 个离散点用拉格朗日插值分式作一个 N 阶多项式 Y(X)，使不在节点上的 X_S，利用插值分式可以求出 $Y(X_S)$的值。

根据计算方法，拉格朗日插值分式为：

$$Y(X) = L_0(X)Y_0 + \cdots + L_N(X)Y_N$$

式中：

$$L_K(X) = \frac{Q(X)}{(X - X_K)Q'(X_k)} \qquad (k = 0,1,\cdots,N)$$

$$Q(X) = (X - X_0)(X - X_1)\cdots(X - X_N)$$

$$Q'(X) = (X_k - X_0)(X_k - X_1)\cdots(X_k - X_{k-1})(X_k - X_{k+1})\cdots(X_k - X_N)$$

在拉格朗日插值中，因用到 N 个节点，每个节点都有两个坐标值，即(X_k,Y_k)，(k=0，1，…，N)，为此，我们必须使用两个一维数组，X(I)，Y(I)分别存放 N 对离散点。程序如下：

```
INPUT PROMPT"N:":N(输入节点数)
DIM X(N), Y(N)
  FOR I=1 TO N
  PRINT "X(";I;"),"Y(";I;")
  INPUT X(I),Y(I)
NEXT I
INPUT PROMPT "XS:";XS ! (输入可求点)
LET Y1=0
FOR I=1 TO N
  LET P=1
    FOR J=1 TO N
      IF I <> J THEN
        LET P=P * (XS-X(J))/(X(I)-X(J))
  END IF
    NEXT J
    LET Y1=Y1+P * Y(I)
  NEXT I
  PRINT "Y1";Y1
  END
```

例 5.29　曲线拟合。在二维的实验中，总可以测得二者相关的一组实验数据。我们想用一个多项式去将这些离散点描绘出来。对于这 N 个离散点，可以用拉格朗日插值去逼近来描绘曲线，阶数必然是 N−1 阶的。由于阶数较高，精度显然不够，为此用曲线拟合进行逼近，效果要好得多。

所谓曲线拟合，就是要找一个低阶多项式来逼近原多项式，并且一定满足在所实验点上的两个函数值的差的平方和为最小，即

$$\sum_{i=1}^{N}[g(X_i)-Y_i]^2$$

式中，$g(X_i)$为拟合曲线在点 X_i 的值，Y_i 为原曲线在 X_i 点的值。曲线拟合的理论可参阅有关计算方法的书籍。程序如下：

```
DIM X(1),Y(1),C1(1,1),C2(1,1),C3(1,1)
INPUT PROMPT"N,M:";N,M
MAT INPUT PROMPT"X(N):";X(N)
MAT INPUT PROMPT"Y(N):";Y(N)
MAT C1=ZER(M+1,N)
FOR I=1 TO M+1
  FOR J=1 TO N
    LET C1(I,J)=X(J)^ (I-1)
  NEXT J
NEXT I
MAT C2=ZER(N,M+1)
MAT C2=TRN(C1)
MAT C3=ZER(M+1,M+1)
MAT C3=C1 * C2
MATC3=INV(C3)
MAT Y=C1 * Y
MAT Y=C3 * Y
FOR I=1 TO M+1
  PRINT"A(";I-1;")=";Y(I)
NEXT I
END
```

小　　结

(1) 数组在使用前必须先用 DIM 语句定义。定义内容包括数组名、维数及每维的上下界。数组中每个维数除 DIM 定义其上下界之外，还可以用 OPTION BASE 语句来定义其下界，缺省的下界为 1。数组的大小随上下界的确定而确定。只可用整常数定义数组大小，不可用变量或表达式。

(2) 数组中元素又称下标变量，数组元素由其下标来区别。下标应括在圆括号中，可以是常量，也可以是变量或表达式。当变量或表达式值非整数时，机器会自动取整。

(3) 一个数组的下标可以用任何整数作为它的上下界。一维数组元素是由下标顺序存放内存，二维数组元素是逐行放内存，三维数组元素是逐页逐行存放在内存的。

(4) MAT INPUT 和 MAT READ 语句可用于读入一维、二维数组元素，其读入顺序和计数器(或里程表)顺序相似，先变低位，后变高位。

(5) MAT INPUT，MAT READ 和 MAT 赋值语句都可以调整数组的上下界，但不能改变数组维数。MAT INPUT Array(?)仅可用于一维数组的上界重新定界。

(6) 矩阵函数为数值计算带来很多方便，用户应充分利用。

(7) 一维数组 A 可以看成是只有一行的二维数组 $A_{1\times N}$。二维数组可看成矩阵，行列数相等的矩阵称为方阵。数组乘法只适用于具有 $A_{M\times L}$ 与 $B_{L\times N}$ 形式的两个矩阵。常数数组 IDN，IDN(X)和矩阵函数 INV(A)，DET(A)，DET 都是对方阵的操作。

习　　题

1. 指出下列数组各定义了多少元素？

```
DIM X(7 TO 40),G(5 TO 15,18 TO 37),
DIM A(5,15 TO 50),B(10,10)
```

2. 改正以下程序中的错误：

```
(1) FOR I=1 TO M
      INPUT M,N
      DIM A(M,N)
      FOR J=1 TO N
      NEXT J
      NEXT I
```

```
(2) INPUT PROMPT"M,N:";M,N
    DIM A(M,N)
    FOR I=1 TO M
      FOR J=1 TO N
        INPUT A(I,J)
```

```
NEXT J
    NEXT I
```

3. 设有两组数，第 1 组为 15,3,7,19,21,6,8,10；第 2 组为 5,2,4,11,1,9,12,17。首先将两组数据读入两个数组 A(8)和 B(8)，并分别对这两个数组的数据排序，然后将两个数组合并成一个数组 C，并将 C 组排序。

4. 有一个二维数组，找出其中值最大的元素以及该元素所在的行和列。

5. 有一个二维数组，求所有靠边元素值之和。

6. 计算下列矩阵：

(1) $\begin{pmatrix} 1 & 6 & 4 \\ -4 & 2 & 8 \end{pmatrix} + \begin{pmatrix} -2 & 0 & 1 \\ 2 & -3 & 4 \end{pmatrix}$

(2) $\begin{bmatrix} 3 & 4 & 2 \\ 4 & 5 & 7 \\ 7 & 2 & 6 \end{bmatrix} - \begin{bmatrix} 2 & 1 & 1 \\ 3 & 1 & 2 \\ 4 & 1 & 5 \end{bmatrix}$

(3) $\begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 6 \\ 3 & 6 & 9 \end{bmatrix} \begin{bmatrix} -1 & -2 & -4 \\ -1 & -2 & -4 \\ 1 & 4 & 4 \end{bmatrix}$

(4) $\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 \end{bmatrix}$

7. 利用矩阵解下列线性方程组：

(1) $\begin{cases} x+2y=1 \\ -x-y+z=2 \\ -2x+2y-z=2 \end{cases}$

(2) $\begin{cases} 2x_1+x_2-5x_3+x_4=8 \\ x_1-3x_2-6x_4=9 \\ 2x_2-x_3+2x_4=-5 \\ x_1+4x_2-7x_3+6x_4=0 \end{cases}$

8. 编一程序，使数组 A 改写为数组 B。

$$A=\begin{pmatrix} 11 & 12 & 13 & 14 & 15 \\ 16 & 17 & 18 & 19 & 20 \end{pmatrix}$$

$$B=\begin{bmatrix} 20 & 15 \\ 19 & 14 \\ 18 & 13 \\ 17 & 12 \\ 16 & 11 \end{bmatrix}$$

9. 设计打印以下报表的程序，其中学号和 3 门功课成绩由 READ/DATA 语句提供，DATA 的数据以学分为序从小到大排列。输出表格中以总分降序排列。总分相同，学号小者在前，但名次相同。

名次	学号	数学	语文	英语	总分	均分
1	9203	90	95	88	273	91
2	9209	75	80	86	241	80.3
3	9201	70	85	62	217	72.3
3	9205	82	91	44	217	72.3
4	9208	68	70	72	210	70
5	9204	50	72	60	182	60.7
学科累计		435	493	412	1340	
学科均分		72.5	82.2	68.7	74.4	

10. 一农民在菜园周围种了 12 棵萝卜，并按顺时针方向给每棵萝卜 1 个代码，若农民从第 1 号开始拔，然后按顺时针方向每隔两棵拔一棵，直到拔完为止，试编一个程序，按拔萝卜顺序打印出萝卜代码的数列。

11. 打印 9×9 的行方阵，使每行每列两对角线的元素和都相等。例如 3 的行方阵为：

$$\begin{pmatrix} 4 & 9 & 2 \\ 3 & 5 & 7 \\ 8 & 1 & 6 \end{pmatrix}$$

9 个元素使用 1～9 共 9 个无重复数。

12. 某单位有 1000 人报名献血，但只要 1 人，现决定：报名者站成一队，从第 1 名开始，第 1 名到第 3 名报数，凡报到了就退出，重复进行，直到第 n 次报数后，规定最后报 3 的人是献血者。问共报了几次数，这位献血者最初站在什么位置？

13. 某单位分配住房按积分排次序，规定工龄分每年 0.5 分；助教、讲师、副教授、教授的职称分分别为 20、25、30、35；科员、科级、处级、局级的职务分分别为 20、25、30、35；祖孙三代照顾 3 分；独生子女照顾 2 分；侨属、港台亲戚照顾 2 分。积分相同，年长者在先。每个人的数据包括编号、姓名、出生年月、职称、职务、祖孙三代、独生子女、侨港台属。编制排队程序，打印出排队表。

第 6 章　函数与子程序

TDP(TRADITIONAL DATA PROCESSING)系统程序代表着系统解决问题所用的知识、方法和策略。其中，问题处理知识和程序实现知识混杂在一起。当程序比较庞大时，程序编写、调试、修改和纠错带来很大的困难。结构化程序设计的主导思想是将复杂问题分解为若干逻辑上独立的小问题，当作一个程序块(模块)单独设计，由于模块逻辑上独立，逻辑关系相对地比较简单，程序设计比较简单，调试纠错容易，可靠性强，模块一旦建立可由程序共享，且可用不同模块搭配以解决复杂问题。结构化程序设计方法提高了程序设计效率。函数与子程序是结构化程序设计技术的两个重要工具。

True BASIC 提供了许多标准函数。标准函数多少是语言系统功能强弱的标志之一，但标准函数的数目毕竟有限。为了提供更多的方便，允许用户自已定义函数。由函数名返回函数调用值。

此外，True BASIC 允许把要重复使用、逻辑上相对独立的程序段编写成子程序。子程序编写一次，可供多次重复调用。

自定义函数和子程序都是逻辑上相对独立的语句模块，不同之处是前者利用函数名返回一个数值或者一个字符串，因此，函数的作用与变量类似，只能在表达式中应用，并根据返回数据类型，有数值函数和字符串函数之分。子程序或者可能只有调用结果，不返回任何数据，或者利用全程变量或参数返回一个或多个数据。

本章首先介绍 Ture BASIC 提供的标准函数中的字符串处理函数、时间和日期函数及标准函数应用。着重介绍自定义函数和子程序的定义和设计，并介绍程序库的建立和使用，及出错处理结构和出错函数。

6.1　标准函数

True BASIC 提供了很多标准函数，如第 3 章中介绍的数值处理函数、USING 函数，第 5 章中介绍的矩阵函数、数组函数和逻辑函数。本节将介绍字符串处理函数、日期和时间函数以及标准函数的应用。

6.1.1　字符串处理标准函数

利用计算机进行数据处理时，要处理的信息除数值外，还有大量非数值数据。这些非数值数据一般都是用字符串表示，所以程序设计言语的字符串处理能力十分重要。True BASIC 提供了丰富的字符串处理标准函数，根据字符串处理功能可分为两类：字符串函数和字符串与数值转换函数。

(1)字符串函数

字符串函数的特点是：自变量是字符串经该函数处理后得到的结果仍是字符串，因此所有字符串函数名皆有 $ 后缀。

①英文字母变换成小写函数

格式： LCASE $ (<字符串表达式>)

功能： 将<字符串表达式>代表的字符串中所有字母变为小写字母返回，除英文字母以外，其它字符不受变换影响，保持不变。

```
OK. PRINT LCASE $ ("* * AFTER CONVERSION * *")
 * * after conversion * *
OK.
```

②英文字母变换成大写函数

格式： UCASE $ (<字符串表达式>)

功能： 将<字符串表达式>代表的字符串中所有字母变为大写字母返回，除英文字母以外，其它字符不受变换影响，保持不变。

```
OK. PRINT UCASE $ ("* * afterconversion * *")
 * * AFTER CONVERSION * *
OK.
```

③删除字符串首部空格函数

格式： LTRIM $ (<字符串表达式>)

功能： 删除<字符串表达式>代表的字符串首部(左端)的所有空格返回。

```
OK. PRINT LTRIM $ ("    COMPUTER    ")& "ROOM"
COMPUTE   RROOM
OK.
```

④删除字符串尾部空格函数

格式： RTRIM $ (<字符串表达式>)

功能： 删除<字符串表达式>代表的字符串尾部(右端)的所有空格返回。

```
OK. PRINT   RTRIM $ ("    COMPUTER    ")& "ROOM"
    COMPUTER ROOM
OK.
```

⑤删除字符串首尾部所有空格函数

格式： TRIM $ (<字符串表达式>)

功能： 该函数包含了 LTRIM $ 和 RTRIM $ 两个函数的功能，删除<字符串表达式>代表的字符串中的首尾部所有空格返回.

```
OK. PRINT   TRIM $ ("    COMPUTER    ")& "ROOM"
COMPUTER ROOM
OK.
```

⑥字符串重复函数

格式： REPEAT $ (<字符串表达式>,N)

功能： 将<字符串表达式>代表的字符串重复 N 次组成新字符串返回。当 N 值有小数时，自动四舍五入取整；当 N>0 时，则把<字符串表达式>代表的字符串重复连接 N 次；当 N<0 时，给出错误信息。

例 6.1　打印某厂一年中每月生产棒形统计图。

```
! exam6—1
! REPEAT $ function
LET star $ =″ *″
FOR i=1 to 12
    READ m $ ,p
    PRINT m $ ; tab(15) ; repeat $ (star $ ,p)
NEXT i
DATA ″January ″ ,11,″ February″ ,12,″ March″ ,15,″ April″ ,17,″ May″ ,18,″ June″ ,20,
DATA″ July″ ,19,″ August″ ,17,″ september″ ,11,″ October″ ,16,″ November″ ,15″ December″ ,12
END
```

……………………………………………………

```
OK. run
January      * * * * * * * * * * *
February     * * * * * * * * * * * *
March        * * * * * * * * * * * * * * *
April        * * * * * * * * * * * * * * * * *
May          * * * * * * * * * * * * * * * * * *
June         * * * * * * * * * * * * * * * * * * * *
July         * * * * * * * * * * * * * * * * * * *
August       * * * * * * * * * * * * * * * * *
September    * * * * * * * * * * *
October      * * * * * * * * * * * * * * * *
November     * * * * * * * * * * * * * * *
December     * * * * * * * * * * * *
ok.
```

(2)字符串与数值转换函数

字符串与数值转换函数包括把字符串转换成数值或把数值转换成字符串。True BASIC 语言中，后者函数名有 $ 后缀，函数返回的是字符串，前者无后缀，返回的是数值。

①字符串转换成数值的函数

(a) 求字符串长度函数

格式： LEN(<字符串表达式>)

功能： 返回<字符串表达式>代表的字符串的字符个数，即字符串长度值。函数返回值为 0～32767 之间的整数。空串的长度为 0。

```
OK. PRINT LEN(″TRUE   BASIC″)
10
OK.
```

(b) 字符转换为 ASCII 码值的函数

格式： ORD(<字符>/<字符名称>)

功能： ORD 函数将返回<字符>或<字符名称>所对应的 ASCII 码值。当 ORD 函数的参数是单个字符时，返回该字符的 ASCII 码值，如字符“A”的 ASCII 码值为 65。ORD 函数的参数可用<字符名称>，例如 ORD(“LCA”)，返回小写字母“a”的 ASCII 码值 97。又如“BS”是控制字符表示退格，ORD(“BS”)返回的 ASCII 码值为 8。除<字符名称>外，若参数

是多个字符时，True BASIC 不能识别时，给出“IMPROPER ORD STRING”错误信息。

```
OK. PRINT ORD("Y")
89
OK.
```

(c) 字符串查找函数

格式： POS(<字符串表达式>,<子串表达式>,<数值表达式>)

功能： 从给定字符串中寻找子串位置，其中<字符串表达式>是给定的字符串，<子串表达式>是要寻找的子串。<数值表达式>表示从字符串中哪一位置开始查找子串，它是一任选项，当<数值表达式>的值小于等于 1 或该项缺省时，表示从第一个字符开始查找。

当<数值表达式>的值为小数时，则自动四舍五入取整。当子串在给定的字符串中找到时，POS 函数返回子串在字符串中第一次出现的起始位置值。如查找不成功，则返回值为 0。

```
OK. PRINT POS("multiplicative table","table",3)
16
OK.
```

(d) 把字符串转换成数值的函数

格式： VAL(<字符串表达式>)

其中<字符串表达式>是数字字符串，可以包括数符+，-及指数符号 e,E。

功能：VAL 函数将一个由数字字符串转换成一个对应的数值返回。非法数作参数时，包括带有单位、“$”符、数位分隔符“,”等，将出现“VAL STRING ISN’ T A PROPER NUMBER.”错误信息。

```
OK. PRINT VAL("-218"), VAL("3.49"&"21"), VAL("2.41E+4")
-218          3.4921          24100
OK. PRINT VAL("34,584,878")
VAL STRING ISN’ T A PROPER NUMBER
OK.
```

②数值转换成字符串的函数

(a) ASCII 码值转换成相应字符的函数

格式： CHR $ (<数值表达式>)

功能： 其中<数值表达式>的值作为 ASCII 码值，CHR $ 函数返回与它所对应的一个字符。True BASIC 采用 256 个符号集，所以<数值表达式>的取值范围是 0～255，当超过该范围时，True BASIC 自动用 256 取模。CHR $ 与 ORD 是互为逆运算函数。

```
OK. PRINT CHR $ (68), CHR $ (326)
D          F
OK.
```

(b) 把数值转换成字符串的函数

格式： STR $ (<数值表达式>)

功能： STR $ 函数将一个数值数据转换成相应的字符串。STR $ 函数与 VAL 函数是互为逆运算函数。

```
OK. PRINT STR $ (1131238), STR $ (2.3E-3), STR $ (123E+12)
```

```
1131238            .0023            1.23e+14
OK.
```

(c) NUM $ (<数表>),NUM(<串表>)

这两个函数专用于记录文件中的数值与数字字串转换，将在第 7 章中介绍。

6.1.2 时间和日期函数

True BASIC 提供了时间和日期函数，这些函数可给出计算机中设置的时间和日期。

(1)时间累计函数 TIME。该函数返回当前时间，当前时间以午夜 0 时开始累计的秒单位总数，它由计算机中时钟实现，时钟在午夜自动清 0。例如午夜过 3 分，则 TIME 的函数值为 180。如果机器上无时钟，则 TIME 将为－1。例如：

```
OK. PRINT TIME
45678.3
OK.
```

(2)时间函数 TIME $ 。该函数返回一个用字符串形式表示的当前时间，时间形式为：“HH::MM:SS”，其中 HH 表示时，MM 表示分，SS 表示秒。午夜时函数值为“00:00:00”。如果机器上无时钟，则 TIME $ 函数返回为“99:99:99”。例如：

```
OK. PRINT TIME $
09:35:08
OK.
```

(3)天数总计函数 DATA。该函数可计算出从当年元旦开始至今的总天数，其形式是“YYDDD”，其中 YY 表示年，DDD 表示这一年的第几天。例如：

```
OK. PRINT DATA
93124
OK.
```

(4)日期函数 DATA $ 。该函数返回当前日期，日期形式为“YYYYMMDD”，其中 YYYY 表示年份，MM 表示月份，DD 表示某月中的第几天。例如：

```
OK. PRINT DATA $
19930504
OK.
```

这一结果表示当前日期是 1993 年 5 月 4 日，若机器上无时钟，则此函数返回 00000000。

6.1.3 标准函数的应用

例 6.2 编写一个程序，它能把 A $ 中的字符按相反的顺序打印出来。

```
LET a $ ="abcdefghijklmnopqrstuvwxyz"
LET c $ =""
FOR i=26 to 1 step－1
    LET b $ =a $ [i:i]
    LET c $ =c $ &b $
NEXT i
PRINT "c $ =";c $
END
..................................
ok. run
c $ =zyxwvutsrqponmlkjihgfedcba
ok.
```

例 6.3 设计一个程序，给一批数码前面加“0”，使得它们的位数与这批数中位数最长的数码位数相同。

解:用数组 A$ 存放这批数码,数组 R 存放各数码的长度,N 表示这批数码的个数,RMAX 为最大数码长度。

```
! Append 0 to the Same bits
DIM a$(1) , r(1)
INPUT n
MAT INPUT a$(n)
MAT r=zer(n)
LET rmax=0
FOR j=1 to n
    LET r(j)=len(a$(j))
    LET rmax=max(rmax,r(j))
NEXT j
FOR i=1 to n
    IF r(i) <> rmax then
         LET y$ =repeat$("0",rmax-r(i))
         LET a$(i)=y$&a$(i)
    END IF
NEXT i
MAT PRINT a$,
END
……………………………………
ok. run
? 3
? 314,4957,8174981
0000314     0004957     8174981
ok.
```

例 6.4　编码和译码程序。编码是按一定规律将输入的字符串转换为新的字符串。编码的方法很多,本题采用的方法是:先取每个字符的 ORD 函数值加 2,再将其和当作 ASCII 码值,由 CHR$ 函数得到一个新的字符作为编码符。逐次将变换了的字符连接,就得到一个新的字符串编码。译码是编码过程的逆变换。

```
! Coded and decoded
INPUT prompt"enter message";m$
FOR k=1 to len(m$)
    LET t$ =m$[k:k]
    LET c=ord(t$)+2
    IF c>255 then LET c=c-255
    LET n$ =n$&chr$(c)
NEXT k
PRINT "the coded message is=";n$
LET l=len(n$)
FOR k=1 to l
    LET t$ =n$[k:k]
    LET c=ord(t$)-2
    IF c<0 then LET c=c+255
    LET e$ =e$&chr$(c)
NEXT k
PRINT "the decoded message is=";e$
END
……………………………………
ok. run
Enter Message ? najing
The coded message is=PCLKPI
The decoded message is=najing
ok.
```

6.2 自定义函数

True BASIC 允许用户根据需要自已定义函数,称自定义函数,简称函数。

用户可以把一个函数式用一个语句或模块定义成函数,当调用函数时,程序转向执行定义语句或模块,函数名返回调用的函数值。根据返回数据的类型可分为数值型和字符串型自定义函数。

自定义函数又根据它在程序中所处的位置不同,分为内部自定义函数和外部自定义函数。

6.2.1 函数的自定义格式

自定义函数有单行自定义函数和多行自定义函数结构两种。

(1)单行自定义函数

格式为：

DEF<函数名[$]>[(<形式参数[{,<形式参数>}])]=<表达式>

(2)多行自定义函数结构

格式为：

DEF　<函数名[$]>[(<形式参数>[{,<形式参数>}])]
　　………………………………
　　………………………………
　　………………………………
　　LET　<函数名[$]>=<表达式>
　　………………………………
　　………………………………
　　………………………………
　　END　DEF

说明：

(1)DEF 为自定义函数的关键字。单行自定义函数必须在一行内写完。多行自定义函数结构以 DEF 语句开始，以 ENDDEF 语句结尾。该函数结构中可以使用任何语句，但至少要有一个给函数名赋值的语句，以便返回函数调用值。表达式的值必须与函数名同一类型。

(2)<函数名>与变量名的命名规则相同。函数允许是数值型或字符串型，但字符串函数名的最后一个字符应是“$”。在同一程序中，用户所定义的函数名尽量不要和其它变量或数组同名，以免混淆。

(3)自定义函数可以带参数，也可以不带参数。对于无参函数，函数名后的括号可以省略。函数参数必须包括在圆括号中，多个参数用逗号分隔，参数命名同变量命名规则。在函数定义中的参数称形式参数，简称形参。它们和变量不同，首先它们不占用存贮空间，它们随函数调用执行而建立，并由调用的实在参数把数据传递给它们，随调用结束，自动撤消；其次，形式参数是 True BASIC 语言中唯一未定义可先出现的变量，又称哑元。

(4)形式参数用来给函数处理传递数据，可以是简单变量、数组。但通道或文件不能用作函数的参数。

(5)函数定义语句或结构位置随意，根据它的位置分内部函数和外部函数，内部函数在调用程序单位内定义(END 语句之前)，外部函数在调用程序单位外(END 之后)定义。

(6)注意：True BASIC 中使用的无参标准函数名不能再定义为自定义函数名，它们是 DATA，DATA $，TIME，TIME $，EXLINE $，EXTEXT $，EXTYPE，MAXNUM，PI，RND，DET 等。

6.2.2　自定义函数的调用格式

自定义函数由程序中表达式调用，作为表达式的一个分量，函数调用遵循原则是先定义或声明后调用，否则出错。一旦定义后，与标准函数一样调用。自定义函数的调用格式为：

<自定义函数名>[(<实在参数>[{,<实在参数>}])]

说明：

(1)内部自定义函数只能在定义它的程序单位中调用。

（2）实在参数和函数的形式参数之间必须保持类型相同，个数一样，位置对应的原则。在调用时，实参数据将一一对应地传递给形参。如果是简单变量形参，则实参可以是常数、表达式、函数名；如果是数组形参，则实参用已定义的同维数组，调用时规定括号内只写数组名，而不必用逗号表示数组的维数。

（3）实参表是用逗号“，”分隔的多个实参，在函数调用时，将实参从左到右逐个求值，并将相应的值赋予各形参命名的临时单元存贮，程序控制转到函数定义模块执行。如果定义模块中某个形参的值被修改，亦不会改变调用程序中相应的实参。实参只给形参传递数据或数组数据，因此在函数中使用的参数称值参。函数调用结果由函数名返回。

（4）内部函数虽然在调用它的程序单位中定义，为使系统不会误解成未定义数组的下标变量，必须遵循先定义后调用的原则，如果调用在前，定义在后，可用 DECLARE 语句特别声明是函数名，否则给出错误信息：“THIS MUST FIRST APPARE IN A DIM OR DECLARE DEF”

6.2.3 函数声明语句

函数声明语句的格式为：

DECLARE　DEF　<函数名[＄]>[{,<函数名[＄]}]

DECLARE 语句声明先使用后定义的函数，已作声明的函数即使后定义，亦可调用；外部函数必须用 DECLARE 语句声明函数，才能调用。

6.2.4 内部函数

内部函数在程序单位内定义，或在一个外部子程序或外部函数中定义，内部函数和所在程序单位共享变量。

例 6.5　输出三个数及其中最小的数。

```
! print three numbers & smallest
DEF MINN(a,b,c)=MIN(MIN(a,b),c)
INPUT PROMPT "Enter x,y,z=":x,y,z
PRINT "x","y","z"," smallest"
PRINT x,y,z,minn(x,y,z)
end
```

```
.................................
ok. run
Enter x,y,z=45,38,97
x      y      z      smallest
45     38     97     38
ok.
```

说明：程序定义了单行内部函数 MINN(a,b,c)，其中 a,b,c 是形参，x,y,z 是实参。当调用时，输入的三个数通过实参 x,y,z 传递给形参 a,b,c。由内部函数 MINN 求得其中最小数 smallest。

例 6.6　按下列要求建立考试成绩转换为等级评价的内部函数，调用它输出 5 个成绩的评价：

90=<S<=100	显示"EXCELLENT"
80=<S<=89	显示"GOOD"
60=<S<=79	显示"NORMAL"
S<=59	显示"DOWN"

```
! convert from score to grade
DEF grade $ (s)
    SELECT CASE s
        CASE is>=90
            LET grade $ ="excellent"
        CASE 80 to 90
        LET grade $ =″good″
    CASE 60 to 79
        LET grade $ =″nomal″
    CASE is < 60
        LET grade $ =″down″
    END SELECT
    LET s=s+1
END DEF
PRINT ″grade″, ″score″
PRINT
DO  while more data
    READ s
    PRINT grade $ (s),s
LOOP
DATA 58,99,30,72,85
END
............................
ok. run
grade        score

down         58
excellent    99
down         30
nomal        72
good         85
ok.
```

程序中从 DEF GRADE $ (S)到 END DEF 为一个内部多行自定义函数，它将调用传递给形参 S 的数值作为分数，转换成文字等级返回。PRINT 语句调用内部函数 GRADE $ (S)，将分数转换成等级输出，其中实参 S 是从数据表中读取的考试分数，DO－LOOP 循环由逻辑函数 MORE DATA 控制，直到数据全部读入转换后，结束程序运行。

函数定义中的形式参数 S 不占内存单元，只有在函数调用时，才临时开辟一个用形参命名的内存单元，存放对应实参传递赋给的数据，调用结束，撤消临时单元，归还所占内存单元。所以形式参数只在其定义的函数模块中有效，具有局部性。在本例中形参 S 和实参 S 同名，但互不相干。函数调用过程中形参 S 变化，不影响实参 S。为说明形参 S 的局部性，在函数定义中有意修改形参 S=S+1，调用返回后，程序再输出 S，从运行结果明显可见并不影响程序中的 S 值。

例 6.7　用数组作参数。

```
! parameter of array
DEF   a(b())
      LET s=0
      FOR i=1 to UBOUND (b)
          LETs=s+b(i) * b(i)
      NEXT i
      LET a=s
END DEF
DIM c(6),d(6)
MAT c=con (UBOUND (c))
MAT d=2 * c
PRINT a(c),a(d)
END
...............................
ok. run
  6            24
ok.
```

数组作形式参数时，由数组名(无须 DIM 定义)和圆括号逗点组成，逗点表示维数，例如一维数组 A()，二维数组 B(,)，三维数组 C(,,)等。

本例 DEF a(b())语句中，a 为函数名，b()为数组形式参数。语句 PRINT a(c)，a(d)中的 c 和 d 为实在参数，它们是程序中已经定义的两个一维数组。函数被调用时，将整个数组传送到临时数组中，因此运行速度慢，占用内存单元较多。

例 6.8　输入一个英文句子，判断是否是 PALINDROME，即向后或向前拼皆为一样的字符串。

```
! pal indrome
DEF conv $ (r $ )
```

```
    LET c$=""
    FOR i=len(r$) to 1   step -1
        LET  c$=c$ &  r$[i;i]
    NEXT i
   LET conv$=c$
END DEF
DO
  INPUT prompt "Enter a string?" ;a$
  IF a$="." then EXIT DO
  IF len(a$)<>0 then LET t$=conv$(a$)
  IF a$=t$ then
       PRINT t$;" is palindrom."
   ELSE
          PRINT t$;" non-palindrom."
      END IF
LOOP
END
.......................................................
ok. run
Enter a string? madam
madam is palindrom
Enter a string? nrettap
nrettap non-palindrom
Enter a string?.
ok.
```

说明：程序定义了 CONV$ 多行结构的内部函数，它将 R$ 字符串中每个字符前后转置返回，程序 A$ 接受键入字符串，它作实参调用 CONV$ 函数，并与返回结果比较，判断并显示输入的字符串是否前后对称。

例 6.9　在函数定义模块中，可以有一条或若干条 EXIT DEF 语句，当调用函数时，执行到该语句，则调用函数结束，返回调用它的程序处。

```
! exit def
DIM a(10),b(10)
DEF dd$(a(),x)
    LET dd$="no find"
    FOR i=1 to UBOUND(a)
      IF a(i)=x then
        LET dd$="find"
        EXIT DEF
      END IF
    NEXT i
END DEF
INPUT prompt"y=";y
MAT READ b
PRINT y,dd$(b,y)
DATA 51,69,4,1,31,71,16,27,9,17
END
...................................
ok. run
y=16
16          find
ok. run
y=88
88          no find
ok.
```

6.2.5　外部自定义函数

外部函数与内部函数不同的是，外部函数在调用它的程序单位的 END 语句之后定义，也可以是库文件中的函数；外部函数是独立的程序单位，定义中使用的变量是局限于该函数定义结构中有效的局部变量。每次调用时，所有数变量都被初始化为 0，串变量初始化为空串，调用结束自动撤消。

所谓程序单位是指从程序开始到 END 之间的一个语句模块，或者一个外部函数，或者一个外部子程序。内部函数与调用它的程序共同构成一个程序单位。外部函数使用之前，需要用 DECLARE DEF 语句说明。在这个语句中仅需列出外部函数的名字，不必声明它有多少个参数。

例 6.10　一个外部函数的例子。

```
! external function
DECLARE DEF g$
```

```
INPUT prompt"enter a character:";x$
INPUT prompt"how lines?";m
FOR i=1 to m
    PRINT tab(m-i+1);g$(x$,2*i-1)
NEXT i
END
DEF g$(a$,n)
    LET r$=""
    FOR j=1 to n
      LET r$=r$ & a$
    NEXT j
    LET g$=r$
END DEF
………………………………
ok. run
Enter a character:h
How lines? 5
             h
            h h h
           h h h h h
          h h h h h h h
         h h h h h h h h h
ok.
```

函数 G$定义在其调用程序的 END 之外，它是一个外部函数，是一个独立的程序单位。外部函数中的变量都是局部变量。

6.2.6 变量和参数的作用域

(1)全程变量和局部变量

全程变量是用在一个程序单位中任何地方都有效的变量，在程序单位中以及它所包含的内部函数和内部子程序中定义的变量均为全程变量，被整个程序单位共享。而外部函数和外部子程序中定义的变量均是局部于该函数或子程序中有效的变量，称局部变量，它在调用程序中无效。

例 6.11 全程变量和局部变量举例。

```
! whole journey' veriation and part veriation
DEF y(x)
    SELECT CASE x
      CASE is>0
        LET yy=10
    CASE is=0
        LET yy=0
    CASE is<0
        LET yy=-10
    END SELECT
    LET y=yy/n
END DEF
LET n=5
PRINT y(3)
END
……………………………
ok. run
2
ok.
```

程序中函数 Y(X)是内部函数，它与调用程序共同构成一个程序单位，变量 N 在调用程序中定义，它在函数定义结构中和调用程序中都出现，是同一变量，在整个程序单位内共享，所以 N 是全程变量。

在一个程序单位中，共享变量用全程变量，在程序中的程序模块之间传递数据和处理结果，使用得当时是十分有用的。正因变量有全程特性，程序和程序单位中各模块设计制约性强，必须从全局考虑，稍有考虑不周，可能引起变量冲突，使程序运行得不到预想的结果，由于变量冲突的错误的隐蔽性，查错、纠错都比较困难。

例 6.12 变量冲突问题。

程序 1:一个有错误的程序。

```
! an insidious bug
```

程序 2:改写过的正确程序。

```
! using an external function
```

```
DEF p$(n)
    LET s$=""
    FOR i=1 to n
      LET s$=s$&"x"
    NEXT i
    LET p$=s$
END DEF
FOR i=1 to 4
    PRINT"HOW many x' s:";
    INPUT n
    PRINT p$(n)
NEXT i
END

…………………………………………
ok. run
How many x' s:5
xxxxx
ok.
```

```
DECLARE DEF p$
FOR i=1 to 4
    PRINT"How many x' s:";
    INPUT n
    PRINT p$(n)
NEXT i
END
DEF p$(n)
    LET s$=""
    FOR i=1 to n
        LET s$=s$&"x"
    NEXT i
    LET p$=s$
END DEF
…………………………………………
ok. run
How many x' s:5
xxxxx
xxxxx
xxxxx
xxxxx
ok.
```

程序 1 希望输出 4 行“X”字符组成的字符串，字符串长度由用户输入确定是 5。但程序只循环 1 次，只输出第 1 行希望长度的“X”字符串，说明其中有错误。

单独从调用程序分析：程序要输出 4 行“X”字符串，用 FOR－NEXT 计数循环结构，循环次数设置为 4 次，每作一次循环，在程序询问“HOWMANYX' S”时，用户回答运行输出“X”的个数，例如第一次是 5，5 作为实参调用函数 P$，输出 P$ 返回由 5 个“X”组成的串，直到循环突破，输出 4 行“X”字符串为止，程序是正确的。

再从 P$ 函数结构进行分析。函数由形参 N 获得一行字符串“X”的个数，比如第一次是 5，也由 FOR－NEXT 计数循环结构控制循环 5 次，每次连接一个“X”字符，直到循环结束，由 5 个 X 组成字符串，经函数名 P$ 返回，函数结构也是正确的。

运行结果出错，原因何在？问题是调用程序和函数定义中循环控制变量都是 I，而函数和调用程序是一个程序单位，I 是一个变量。既用于输出字符串行控制，又用作“X”字符个数控制而造成的错误。程序运行时，I＝1，用户输入 N＝5，调用 P$ 函数，控制执行 P$ 函数，此时，I 又被置成初值 1，当完成 5 个“X”字符连接时 I＝6，返回时 5 个“X”的字符串被程序输出，接着执行程序中 NEXT 语句，它又被修改成 7，返回 FOR 语句 I 值已经超出终值，终止循环和程序运行，结果只输出一行 5 个“X”的字符串，丢失了其它 3 行。

解决方法，一是分别设置两个不同的循环控制变量；二是将内部函数 P$ 改为外部函数，由于外部函数和调用程序一样是独立的程序单位，虽然两个程序单位皆用同名循环控制变量 I，但它们是互不相干的两个不同变量，各有各的控制功能，解决了变量冲突问题。

程序 2 变动了两个地方：第一，把 P$ 函数定义成外部函数，移到了 END 语句之后；第

二，增加了 DECLARE DEF P$ 语句，声明 P$ 为外部函数。

(2)变量的作用域。是该变量所在的程序单位，在程序单位内所有的模块共享变量。而不同程序单位之间的变量毫无关系。变量的生命期同其定义的程序单位。

例 6.13　变量的作用域举例。

```
! action region of veriation
DECLARE DEF sum1,sum2
LET b=3
PRINT "a=";a, "b=";b
PRINT "sum1="; sum1, "a=";a
PRINT "sum2="; sum2, "a=";a,"c=";c
DEF sum1
        LET a=2
        LET sum1=a+b
END   DEF
END
DEF sum2
      LET c,a=3
      LET sum2=a+b+c
END DEF
.........................................
ok. run
a=0           b=3
sum1=5        a=2
sum2=6        a=2           c=0
ok.
```

SUM1 是内部函数，在程序单位内变量共享，主程序对变量 b 已赋过值 3，故 SUM1 的值为 5，变量 a 被改为 2。SUM2 是外部函数，与调用程序变量不公用。故 SUM2 的值为 6，a 为保持原来的值 2，c 在主程序中未定义，为 0。

(3)参数的作用域。是该函数的定义模块，即形式参数是局部变量，在定义结构外即使使用与形参同名的变量，两者之间也毫无关系，形式参数的生命期同定义模块，即在模块调用时建立，调用结束撤消。

例 6.14　参数的作用域举例。

```
! action region of parameters
DEF a(m,n)
     LET m=5
     LET a=m*m*m-n*n
END DEF
LET n=3
FOR m=1 to 3
     PRINT n,a(n,m),m
NEXT m
END
.......................................
ok. run
3           124           1
3           121           2
3           116           3
ok.
```

6.2.7　函数递归

函数的定义中调用定义函数本身，即在函数的定义中又出现调用它自已，称函数递归。

例 6.15　用递归函数的方法求 N!

解：用迭代法可求出 N! =N(N-1)! 因此可得阶乘函数公式：

$$\mathrm{FACT}(N)=\begin{cases}1 & (N=0)\\ N*\mathrm{FACT}(N-1) & (N>0)\end{cases}$$

```
! recursive definition of factorial
DECLARE DEF fact
FOR i=1 to 8
     PRINT i, fact(i)
NEXT i
END
DEF fact(n)
     IF n<=1 then
          LET fact=1
     ELSE
```

```
        LET fact=n * fact(n-1)
    END IF
END DEF
.................................
ok. run
  1          1
  2          2
  3          6
  4          24
  5          120
  6          720
  7          5040
  8          40320
ok.
```

现以 N=4 为例，说明递归过程：

FACT(4)=4 * FACT(3)=4 * 3 * FACT(2)=4 * 3 * 2 * FACT(1)

当执行 FACT(4)～FACT(2)时，都要在 ELSE 块中进一步调用下层。在调用 FACT(1)时，则执行 THEN 块，达到边界条件，即递归的终止条件。接着进入返回过程，由终止条件给出的特定解，按照原来的路径返回。返回到 FACT(2)的 ELSE 块，再返回到 FACT(3)的 ELSE 块，最后返回到 FACT(4)的 ELSE 块，求得 FACT 函数值。其调用和返回的过程如图 6.1 所示。

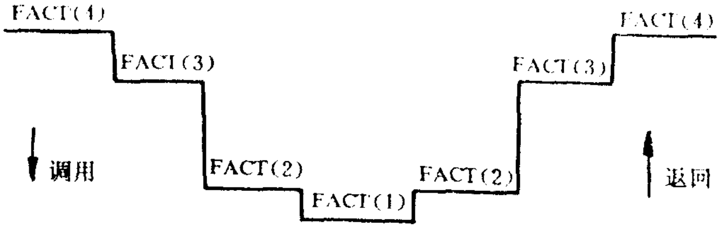


图 6.1　递归过程

递归调用在程序设计中是十分有用的。递归算法的描述简明，可读性好，容易理解，许多看起来相当复杂的问题采用递归后就变得比较容易。但递归程序每层调用都要进行参数替换、现场及断点保护等操作，效率低，占用存储空间多。本例的终止条件是 FACT(0)=1。

6.3　子程序

自定义函数虽然是一个独立的功能模块，但函数的应用有一定的局限性。如自定义函数只能通过函数名返回一个函数值，只能在表达式中使用，在实际问题中可能要求模块调用只有某种效果，不一定产生数据结果，有时又可能要求返回多个数据结果。为此，True BASIC 提供了另一种独立的功能模块结构，这就是子程序。

子程序也是结构化程序设计中的一个模块，按其是否位于调用它的程序单位之内而分成内部子程序和外部子程序两种。

6.3.1　子程序的定义结构

子程序的定义格式为：

```
SUB  <子程序名>[(<形式参数 1>,<形式参数 2>......)]
          .................................. ┐
          .................................. │
              [EXIT SUB]                      ├语句块
          .................................. │
          .................................. ┘
END SUB
```

说明：

(1)<子程序名>的命名规则和变量名相同，由于子程序可能不产生任何数据，或可能产生多个不同或相同类型的数据结果，所以子程序名只是某个子程序的标识符。为避免混淆，使用时尽量不要与调用它的程序单位中的变量、函数、数组重名。

(2)子程序可以不带形式参数，亦可有形式参数，根据需要决定。简单变量、数组和通道都可以作子程序的形式参数，由形参接受数据、数组、文件或图形窗口给子程序处理。

和函数中形式参数类似，子程序的形式参数亦具有局部性，不同的是函数参数只用作单向接受处理数据；子程序参数不仅可接受处理数据，而且可用作返回子程序处理结果。

(3)EXIT SUB 语句可以出现在子程序中的任何地方，当执行到该语句时，程序直接跳到 END SUB 语句，并将控制返回调用程序中，执行 CALL 语句的后续语句。

6.3.2 子程序的调用语句

格式：CALL<子程序名>[(<实参 1>,<实参 2>......)]

说明：

(1)<子程序名>事先无需声明，也不必遵守先定义后调用的原则。<子程序名>由 CALL 语句识别。一般将调用子程序的程序称主程序，主程序用 CALL 语句调用子程序。

(2)在主程序执行到 CALL 语句时，控制转向 CALL 语句指定的子程序，并保存 CALL 语句下一语句地址，执行子程序中的语句，直到遇到一个 EXIT SUB 语句或 END SUB 语句，然后返回到主程序中的 CALL 语句的下一条语句继续往下执行。

(3)CALL 语句中的实参和子程序定义中的形式参数个数应相等，次序应相同，类型一致。常量、函数、表达式、变量、数组和通道都可用作实参。

(4)子程序亦分为内部和外部两种，内部子程序在主程序单位中定义，外部子程序是独立的程序单位，可以是主程序外定义或是程序库文件中子程序。库文件中子程序使用见 6.4 节。

6.3.3 内部子程序

内部子程序和程序单位共享变量，因此主程序可以利用全程变量将数据传递给子程序处理，内部子程序的处理结果同样可利用全程变量回送给主程序。内部子程序可以定义变量，但它亦是整个程序单位有效的全程变量。

例 6.16 分别用无参子程序和带参子程序的方法，编制打印 4 行"*"的程序。

程序 1：无参子程序

```
! internal sub with non—parameter
SUB stars
    PRINT REPEAT$ ("*",9)
END SUB
FOR i=1 to 4
    CALL stars
NEXT i
END

................................
ok. run
```

程序 2：带参子程序

```
! internal sub with parameter
SUB stars(n)
    PRINT REPEAT$ ("*",n)
END SUB
INPUT prompt "x=": x
FOR i=1 to 4
    CALL stars(x)
NEXT i
END
......................................
ok. run
```

```
* * * * * * * * *
* * * * * * * * *
* * * * * * * * *
* * * * * * * * *
ok.
```

```
x=7
 * * * * * * *
 * * * * * * *
 * * * * * * *
 * * * * * * *
ok.
```

说明：

(1)程序 1 中 STARS 是无参的内部子程序。子程序定义由 SUB 关键字开头，以 END SUB 结束。主程序每调用一次 STARS 子程序，就显示一行 9 颗“＊”，共执行 4 次 CALL STARS 语句，所以显示 4 行 9 颗“＊”组成的字符串。

(2)程序 2 中的 STARS(N)为带参数的内部子程序，N 为形式参数，主程序 CALL STARS(X)语句中的 X 为实在参数。使用带参数的子程序更灵活、方便。

例 6.17　内部子程序中的变量是全程变量

```
! internal subroutine transmit data and result
! with whole journey variation
SUB a
    LET x=2*x
    PRINT "x=";x,
    LET y=2
END SUB
LET x=5
CALL a
PRINT "y=";y
END
......................................
ok. run
x=10              y=2
ok.
```

说明：子程序 a 是无参的内部子程序，X，Y 均为全程变量，在整个程序中有效。无参的内部子程序可利用全程变量传递数据和结果。

内部子程序虽然可以通过全程变量传递数据和结果，但带参数的内部子程序结构独立，清晰，更为实用。

6.3.4 外部子程序

外部子程序是定义在调用它的程序单位之外(即 END 之后)，或定义在程序库文件中。

外部子程序中使用的变量是局部变量，每当外部子程序被调用时，变量重新建立和初始化(数变量为 0，串变量为空串)，随外部子程序调用完成而撤消，归还内存单元。所以外部子程序设计有独立性，即使与调用程序有同名变量亦互不干扰，不会发生变量冲突的错误。

外部子程序可以用局部通道，处理文件数据或图形窗口，在外部子程序中打开的所有通道，在子程序返回时自动关闭。

例 6.18　一个无参外部子程序的例子。

```
! external subroutine
LET x=5
CALL a
PRINT"Y="; Y
END
SUB a
    LET x=2*x
    PRINT"X="; X,
    LET y=2
END SUB
.................................
ok. run
x=0               y=0
ok.
```

说明：本例的 A 是无参外部子程序，例中有两个程序单位：主程序和子程序 A，它们有

两个同名变量 X,Y,但它们互不相同,互不干扰,在各自程序单位有效。可见无参外部子程序只能做某个固定的事,每次调用效果相同,没有数据返回,用途不大。但外部子程序的独立性,正如在外部函数中一样(例 6.12),可以避免大程序设计中模块间的变量冲突问题,对结构化程序设计十分有意义。

例 6.19　比较下列两个程序,观察 RESTORE 语句的作用。

程序 1

```
! internal subroutine
SUB gj
print "in-subroutine";
    FOR i=1 to 2
      READ a
      RESTORE
      PRINT a;
    NEXT i
    PRINT
    DATA 8,9
END SUB
PRINT "main:";
DATA 4,5
FOR i=1 to 2
    READ b
    RESTORE
    PRINT b;
NEXT i
PRINT
CALL gj
READ b
CALL gj
END
............................
ok. run
main: 8  8
in-subroutine:8  8
in-subroutine:9  8
```

程序 2

```
! external subroutin
DATA 4,5
PRINT "main:";
FOR i=1 to 2
    READ b
    RESTORE
    PRINT b;
NEXT i
PRINT
CALL gj
READ b
CALL gj
END
SUB gj
    PRINT "ex-subroutine:";
    FOR i=1 to 2
      READ a
      RESTORE
      RINT a;
    NEXT i
    PRINT
    DATA 8,9
END SUB
....................................
ok. run
main:4  4
ex-subroutine:8  8
ex-subroutine:8  8
```

程序 1 中定义的是内部子程序 GJ,整个程序是一个程序单位,程序中所有的 DATA 语句按规则建立一个数据表,由程序单位中各模块共享,因此主程序或子程序的 RESTOR 语句使指针指向本程序单位的数据区的第一个数据 8 上。

第一次调用内部子程序 GJ 后,再作一次重读,使指针指向第二个数据 9,再次调用 GJ,输出是 9,8,说明程序单位中不仅数据表共享,数据表指针亦由该程序单位共同控制。

程序 2 中的 GJ 定义的是外部子程序,它包含了两个程序单位,即主程序和外部子程序。主程序中由主程序 DATA 语句建立数据表,由主程序使用控制数据表指针移动,在调用外部子程序 GJ 时,GJ 程序单位中的 DATA 语句亦随之建立一个临时的数据表,指针指向

表的第一个数据，由 GJ 读取使用控制指针移动，调用结束自动撤消。因此它有作用域和生命期不同的两个局部数据表。

两个程序单位中的 RESTOR 语句只恢复各自程序单位的数据指针，即主程序中 RESTOR 语句使指针指向数据 4，子程序中的 RESTOR 语句使指针指向 8。因此程序 2 的运行结果不同于程序 1。

6.3.5 带参数子程序

无参内部子程序可以通过全程变量传递数据和结果，只要注意不发生变量冲突。无参的外部子程无法传递数据。对无数据提供的程序它只能做同样的工作，与外部函数不同，函数可以利用函数名传递处理结果，但无参外部子程序却无法将处理结果回送主程序，因此无参外部子程序只适用于完成某种调用效果而不需输入数据、不要求返回结果的場合。

如何解决外部子程序的数据传递和结果返回主程序呢？可以利用参数。参数是主程序和子程序的接口，在主程序和子程序间起数据（或结果）的传递作用。

（1）数据参数和变量参数

函数和子程序都允许带有参数，函数参数的作用仅仅是给函数传递数据，实参可以是表达式或数组，函数调用时，建立形参命名的临时变量，将表达式的值、数据或整个数组拷贝到其中。在函数处理过程中，形参值的改变并不影响实参，所以称值参。但子程序中除值参外如果需要回送子程序处理结果，还需有变量参数。变量参数的实参必定是变量，在子程序调用时，它建立地址联系，直接使用实参变量，所以它不仅可以将实参变量中的数据传给子程序，而且在子程序处理过程中，形式参数的值被修改时，则相应的实参变量也被修改，以达到传递子程序处理结果数据的目的，所以称变量参数。调用结束，无论临时变量或变量参数的地址联系全部撤消。实际上，只要实参不是常量、函数和表达式，无论参数在子程序中是否被改变，总是用变量参数处理。例如函数和子程序都可以使用数组作参数，函数将数组按值参处理，每次函数调用时，整个数组被拷贝到一个临时数组中，但子程序将数组参数作数组变量参数处理，和函数处理方法相比，不仅速度快，而且占用存贮空间少。

带参数的子程序调用时，实参应与形参要求相应参数个数、顺序和类型一致。

例 6.20 求 $S=\sum_{i=1}^{n}=I^2$

```
! external subroutine with parameter
INPUT prompt"n=": n
CALL pb(n,t)
PRINT "t=";t
END
SUB pb(k,f)
    LET f=0
    FOR i=1 to k
        LET f=f+i*i
      NEXT i
END SUB
……………………………………
ok. run
n=4
t=30
ok.
```

外部子程序 pb(k,f)中的 k 是数值参数，接受主程序中由对应的实参 n 传递来的数值 4，而 f 是变量参数，把调用子程序的结果返回给主程序中的实参 t。

例 6.21 演示变量参数的数据传递作用。

```
! internal subroutine
SUB f(a,b,c)
    LET c=3*a+4*b
    PRINT a,b,c
    LET a=a-1
    LET b=b-1
END SUB
    LET m=6
    LET n=5
    CALL f(m,n,s)
PRINT m,n,s
```

```
END
.....................................
ok. run
  6          5          38
  5          4          38
ok.
```

例中子程序 f 的 a,b,c 为形参,而 CALL f(m,n,s)语句中的 m,n,s 为实参。在执行 CALL 语句调用子程序时,m,n,s 的值(6,5,0)传递给 a,b,c。子程序计算得 c 值,显示 a,b,c 值为 6,5,38。在子程序执行过程中,a,b,c 值都发生了变化,返回时传递给 m,n,s,所以显示 m,n,s 的值为 5,4,38。

例 6.22　观察调用子程序前后的形参、实参和表达式的值。

```
! with constant and expresion as parameter
SUB gg1(x,y,z)
    PRINT"X="; X, "y="; y, "z=";z
    LET x=x+1
    LET y=y+2
    LET z=z+3
    PRINT"X=" ;X,"y=";y,"z=";z
END SUB
SUB gg2(x$,y$,z$)
    PRINT "X$=";X$,"y$=";y$,
    PRINT "z$=";z$
    LET x$=x$&x$
    LET y$=y$&y$
    LEz$=z$&z$
    PRINT "X$=";X$,"y$=";y$,
    PRINT "z$=";z$
END SUB
LET a=5
LET b=10
CALL gg1(2,2+a,b)
PRINT "a=";a,"b=";b
LET u$="cd"
LET v$="efg"
CALL gg2("xy",u$&"pg",v$[3:3])
PRINT "u$=";u$,"v$=";v$
END
```

```
..............................................
ok. run
x=2          y=7          z=10
x=3          y=9          z=13
a=5          b=13
x$=xy        y$=cdpq        z$=g
x$=xyxy      y$=cdpqcdpq    z$=gg
u$=cd       v$=efg
ok.
```

(2)数组参数。数组作参数时,应使用括号和逗号来标明数组的维数。调用语句中的实在参数只写数组名,数组的维数,各维的大小则由主程序定义。

例 6.23　打印随机矩阵。

```
! ARRAY group as parameter
DIM a(5,5), b(4,3)
CALL gm(a)
CALL gm(b)
MAT PRINT a
MAT PRINT b
END
SUB gm(m(,))
    RANDOMIZE
    FOR i=Lbound(m,1) to Ubound(m,1)
      FOR j=Lbound(m,2) to Ubound(m,2)
        LET m(i,j)=int(rnd * 100)
      NEXT j
    NEXT i
END SUB
```

```
............................
ok. run
```

```
13    86    19    33    7
87    78    31    62    97
45    75    32    78    90
87    6     42    99    16
92    64    55    40    36
```

```
93    48    77
72    32    82
33    29    90
0     20    81
ok.
```

(3)通道参数。子程序可以用通道作为参数,通道号要以“#”号后跟一个1~999之间的任意整常数,被调用时,用实在参数的通道号代替。

例6.24　通道号作参数。

```
! passagen umber as parameter
DATA 4,5
INPUT prompt "display(0) or print(1-999):";o
IF o<>0 then OPEN #o:printer
PRINT #o :"main:";
FOR i=1 to 2
    READ b
    RESTORE
    PRINT #o:b;
NEXT i
PRINT #o
CALL gj(#o)
READ b
CALL gj(#o)
IF o<>0 then CLOSE #o
END
SUB gj(#9)
    PRINT #9:" ex-subroutine:";
    FOR i=1 to 2
      READ a
      RESTORE
      PRINT #9:a;
    NEXT i
    PRINT #9
    DATA 8,9
END SUB
```

(4)子程序与函数在参数传递上的区别

①在函数调用中只将实际参数的数值单向地传送给形参,函数中的操作,不会影响实在参数的值。而函数调用的结果,仅由函数名返回函数值。请看下面的例子。

例6.25　比较函数调用与子程序调用时参数传递的区别。

程序1:函数调用

```
! call function
DEF a(x,y)
  DO
     LET aa=aa+x*y
     LET y=y-1
   LOOP until y<=0
   LET a=aa
END DEF
INPUT prompt "m,n=";m,n
PRINT m,n,a(m,n)
END
..................................
ok. run
m,n=4,2
4       2       12
ok.
```

程序2:子程序调用

```
! call subroutine
SUB a(x,y,z)
  DO
     LET z=z+x*y
     LET y=y-1
   LOOP until y<=0
END SUB
INPUT prompt"m,n=";m,n
CALL a(m,n,t)
PRINT m,n,t
END
..................................
ok. run
m,n=4,2
4       0       12
ok.
```

程序 1 中采用的是函数调用，m 和 n 实参的值向形参 x 和 y 作单向传递，形参 y 的变化不会反过来影响实参 n 的值

②子程序的参数传递与函数不同。在调用子程序时，主程序中的实参变量与子程序中对应的形参变量共用一个存储单元。因此，在子程序执行过程中，如果形参的值发生变化，则主程序中实参的值也随之变化，这样就能达到把子程序中形参的值返回给主程序中的实参。正如本例程序 2 中，调用子程序后，y,z 的值发生了变化，返回主程序时，形参 y,z 的值就传递给了实参 n 和 t。

如果实在参数不是变量，而是常数或表达式，则象对函数那样把常数和表达式的值传送给一个临时变量存起来，然后再把它的值传送到子程序，因此在这种情况下，子程序中形参值的变化，不会影响主程序中表达式的值。

例 6.26　观察参数传递情况。

```
! transmit parameter
SUB f(a,b,c)
    LET c=a*a-b*b
    PRINT a,b,c
END SUB
LET m=5
LET n=4
CALL f(m,n,t)
PRINT m,n,t
CALL f(m-1,n-1,t)
PRINT m,n,t
END
```

```
..................................
ok. run
5          4          9
5          4          9
4          3          7
5          4          7
ok.
```

执行 CALL f(m,n,t)时，实参 m,n 的值传给形参 a,b,同时把 c 的值返回给 t。当执行 CALL f(m－1,n－1,t)时，是把 m－1 和 n－1 的值传给 a 和 b，所以 a＝4,b＝3,c 的值为 7 返回给 t，而 m 和 n 的值并没有改变。

6.3.6　变量的作用域

内部子程序中使用的变量是全局变量，它的有效作用域是包含内部子程序在内的整个程序单位，生命期同程序单位，外部子程序中使用的变量是局部变量，它的作用域只限于定义它的外部子程序，它的生命期同外部子程序，它随外部子程序调用而建立，随调用结束而撤消。

6.3.7　子程序的递归调用

与函数递归类似，如果在子程序的定义中又出现调用本子程序自已，则称其为子程序递归。外部子程序的参数和所有变量都是该程序单位的局部变量，因此外部子程序可递归调用。下面举两个例子。

例 6.27　求两个数的最大公约数。

解　设两个数为 m,n，当 m 除以 n 时得余数 b，如果余数 b 为 0，则除数就是两个数中的最大公约数，如果余数不为 0，则以除数作为被除数，以余数作为除数，再进行上述过程，直到余数为 0 为止。

```
! recurrent call subroutine
INPUT prompt"m,n=";m,n
CALL gcd(m,n,g)
PRINT g
END
SUB gcd(a,b,e)
```

```
  LET e=mod(a,b)
  IF e=0 then
    LET e=b
    EXIT SUB
  ELSE
    LET a=b
    LET b=e
    CALL gcd(a,b,e)
  END IF
END SUB
```

```
ok. run
m,n=39,9
   3
ok. run
m,n=151,7
   1
ok. run
m,n=154,49
   7
ok.
```

……………………………………

由例可知，子程序递归调用过程中，每调用一次都要生成一组新的参数(a,b,e)，虽然这些参数具有相同的名字，但它们每次代表的含义(值)是不同的。设两个数m=39,n=9，当在主程序中执行CALL gcd(m,n,g)语句时，第一次调用子程序gcd，进入时参数a=39,b=9,e=0，当在子程序中执行LET E=MOD(a,b)语句后，e=3，所以接着执行ELSE的语句块，这时a=9,b=3，在子程序的ELSE语句块中执行CALL gcd语句，即在子程序中又调用本子程序，这就是递归调用，但此时参数a,b,e已是一组新的值，完全不同于第一次了。当第二次进入子程序，执行LET E=MOD(a,b)语句(a=9,b=3)后，e=0，即余数为0，所以执行THEN语句块，求得最大公约数为3，并跳出子程序。

值得注意的是：无论是递归函数还是递归子程序，任何有意义的递归都是有限的，即应有一个递归终止条件，本题中的递归终止条件即是余数等于0(e=0)。

例6.28 递归算法求解汉诺塔问题。相传在古代印度的BRAMAH庙中，有3根插在黄铜板上的宝石柱，在其中一个柱子上放了64个金盘子，大盘在下，小盘在上(如图6.2所示)，称为汉诺塔。和尚们想把这些盘子从一个柱子上移到另一个柱子上，规定每次只允许移动一个，而且小盘子不能放在大盘子之下。移动时可利用这3根柱子中的任何一根。

此问题可以归结于3个子任务来实现：

①把1到(N−1)号圆盘从S柱移到I柱，O柱作辅助存放圆盘用。

②把第N号圆盘(大盘)从S柱移到O柱。

③把1到(N−1)号圆盘从I柱移到O柱，S柱作辅助存放圆盘用。

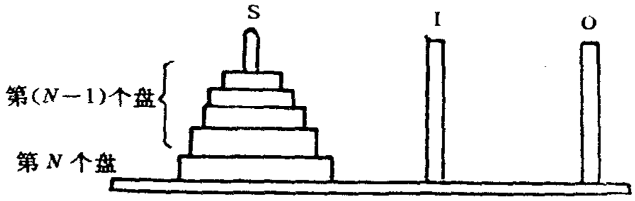


图6.2 汉诺塔

上述过程子任务②把第N个圆盘移动1次，子任务①、③是把(N−1)个圆盘移动2次。现在问题归结为如何将(N−1)个圆盘从I柱移到O柱。与原问题类似，区别是圆盘数变为(N−1)个，且S柱与I柱的作用对调。我们可以重复上述移动方法，递归进行，每次移动的盘子个数减1，直到全部盘子移到O柱上。

因此，HANOI(N,S$,I$,O$)移动过程描述如下：

(1)当N=1时，可直接把圆盘从S柱移到O柱，记作MOVE(S$,N,O$)。

(2)当 N>1 时,首先设法将 S 柱上的(N－1)个圆盘按大小递减的次序,借助 O 柱移到 I 柱上,记作 HANOI(N－1,S＄,O＄,I＄)。将 S 柱上的第 N 个盘(大盘)移到 O 柱,记作 MOVE(S＄,N,O＄)。再设法按大小递减的顺序,将 I 柱上的(N－1)个圆盘借助 S 移到 O 柱,记作 HANOI(N－1,I＄,S＄,O＄)。

执行此时的 HANOI(N－1,I＄,S＄,O＄),又要调用 HANOI(N,S＄,I＄,O＄),只是用(N－1)代替 N,以 I＄代替 S＄,以 S＄代替 I＄即可。显然这是递归调用。

```
! program of hanoi tower problem
! recur subroutine
LET s$ ="S"
LET i$ ="I"
LET o$ ="O"
DO
  INPUT prompt "How many disks?";n
LOOP while n<=0
CALL hanoi(n,s$,i$,o$)
END
SUB hanoi(n,s$,i$,o$)
    IF n=1 then
       PRINT "Move disk";n;"from";s$;
       PRINT "to";o$
    ELSE
       CALL hanoi(n-1,s$,o$,i$)
       PRINT "Movedisk";n;"from";s$;
       PRINT "to";o$
       CALL hanoi(n-1,i$,s$,o$)
    END IF
END SUB
.........................................................
ok. run
HOW   MANY   DISKS  ?   3
MOVE  DISK  1  FROM  S  TO  O
MOVE  DISK  2  FROM  S  TO  I
MOVE  DISK  1  FROM  O  TO  I
MOVE  DISK  3  FROM  S  TO  O
MOVE  DISK  1  FROM  I  TO  S
MOVE  DISK  2  FROM  I  TO  O
MOVE  DISK  1  FROM  S  TO  O
OK.
```

本题的递归终止条件是 N=1。

6.4 程序库文件

位于某个程序单位 END 语句之后的外部函数和外部子程序,只能被该程序单位或该程序单位的其它子程序调用。为了扩大外部函数和外部子程序的使用范围,使不同的程序单位能共享这些外部函数和外部子程序,True BASIC 语言系统提供用户自已建立外部函数、子图、外部子程序库的功能。所谓库文件是存储在磁盘上没有主程序的一些外部函数、外部子图和外部子程序的集合。库文件不能独立运行,但库文件中的函数、子程序或子图可被任何其它程序调用,库文件就象一个“工具箱”,它满足了用户对已通过的程序建立高效程序包的要求。

此外,True BASIC 语言系统有几个函数和子程序库文件,作为软件工具提供用户,其中包含 4 个数学函数库,一个图画子程序库,一个用作菜单设计的子程序库。本节介绍用户程序库的建立、系统程序库的功能及程序库的应用。

6.4.1 用户程序库的建立及应用

外部函数、外部子程序以及第 8 章介绍的外部子图通过参数与调用程序连接,参数是它们的接口,只要接口关系正确,无须了解这些外部模块的具体程序和其中使用的局部变量,允许被任何其它程序调用。为了使不同程序方便地共享这些外部函数、子程序及子图,True

BASIC 允许将它们汇集存储，构成用户自己的程序库文件，称用户库文件。

(1) 用户库的建立。若要将几个外部函数、外部子程序及外部子图汇集构成一个用户库文件，首先将它们的源程序调入内存，在编辑窗口中合并，并在首部增加一个 EXTERNAL 语句行，表明在 EXTERNAL 语句之后是要并入程序库的若干个外部函数、外部子程序或外部子图，然后用系统 SAVE 命令命名，并将它们一起存入磁盘，这样就建立了一个用户程序库文件程序库的结构示意图见图 6.3。

```
EXTERNAL
DEF func1
    ⋮                 外部函数
END DEF
SUB sub1
    ⋮                 外部子程序 1
END SUB
SUB sub2
    ⋮                 外部子程序 2
END SUB
PICTURE p1
    ⋮                 外部子图
END PICTURE
    ⋮
```

图 6.3　库结构的示意图

说明：

①程序库不能以 END 语句结束，由于它是外部函数、子程序或子图的集合，所以它本身不能独立运行，只是供其它程序调用的工具软件。

②同一个程序库文件中所有外部函数、外部子程序、外部子图不能同名。库文件名遵循 MS－DOS 文件命名规则。

例 6.29　把一个外部函数 PO 和一个外部子程序 FACT 组织成一个库，并以“GGLIB”库文件名存盘，其中外部函数 PO 是用来求 X^N，外部子程序 FACT 是用来求阶乘的。

```
! program stock
EXTERNAL
DEF po(x,n)
    LET z=1
    SELECT CASE n
    CASE is>0
        FOR i=1 to n
          LET z=z*x
        NEXT i
    CASE is<0
        LET n=-n
        FOR i=1 to n
          LET z=z/x
        NEXT i
        END SELECT
        LET po=z
END DEF
SUB fact(n,t)
    LET t=1
    IF n<=1 then
      EXIT SUB
    ELSE
```

```
FOR i=1 to n
  LET t=t*i
NEXT i
```

```
END IF
END SUB
ok.save a:gglib
```

GGLIB 是 ASCII 编码的源程序库文件，扩展名是.TRU，亦可以将源程序库文件编译建立编译库文件，扩展名为.TRC，编译库文件不可显示，但占用存贮空间少，运行速度快。

(2)库文件的应用。当某个程序单位需要调用库文件中的函数、子程序或子图时，必须在该程序单位中，调用库文件之前，设置库文件说明语句，该语句格式是：

LIBRARY <库文件名 1> [,<库文件名 2>]……

程序运行时，该语句将相应的库文件调入内存，以备程序使用。

如果从库文件调用某些外部函数，那么除了必须设置库文件说明语句 LIBRARY 外，还需再设置一条函数说明语句：

DECLARE DEF <函数名 1> [,<函数名 2>] ……

声明外部函数名，以便程序调用。

例 6.30　调用例 6.29 建立的库文件中的函数和子程序来计算 cos(x)的值。利用泰勒级数展开式求 cos(x)：

$$\cos(x)=1-x^2/2!\ +x^4/4!\ -x^6/6!\ +X^8/8!\ \cdots$$

其中 x 为弧度。

```
! use program stock
LIBRARY "gglib"
DECLARE DEF po
INPUT prompt"item=": item
INPUT prompt"x=";x
LET j,sum=1
FOR i=2 to item/2 step 2
```

```
   CALL fact(i,t)
   LET y=(-1)*j*po(x,i)/t
   LET sum=sum+y
   LET j=-j
NEXT i
PRINT "x=";x,"cos(";x;")=";sum
END
```

6.4.2 系统程序库文件

除用户程序库文件外，系统还提供几个程序库文件，供程序调用。系统程序库文件的使用方法与用户程序库文件相同。

(1)数学函数库

共有 4 个数学函数库：FNHLIB 是双曲线函数库；FNTLIB 是以弧度为单位的三角函数库；FNTDLIB 是以度为单位的三角函数库；FNMLIB 是数学函数库。这些库以编译形式存储在 True BASIC 系统盘上。表 6.1～6.3 是各库文件的函数名及功能表。

FNTLIB 和 FNTDLIB 两个三角函数库的功能完全相同，使用时取决于角度采用单位。语言系统在缺省时，角度单位是弧度，三角函数库 FNTLIB 有效。程序中可以使用如下语句来指定角度(使用单位为度)：OPTION ANGLE DEGREE。此后只能调用 FNTDLIB 这个库中的三角函数，因为该库中角度单位是度。

数学函数库文件作为数值运算的工具被程序调用，但它们和机内标准函数不同。程序调用数学函数库中的函数前必须指明使用的库文件，将相应库文件调入内存，并声明库中的函数是外部函数，此后才能和标准函数一样使用。为此，每个数学函数库都备有一个未编译的包含文件。在程序编辑时，只要在背景窗的命令行键入：

OK. INCLUDE <包含文件名>

表 6.1　FNHLIB.TRC(双曲线函数库表)

包含文件	函　数	说　明
FNH.TRU	SINH(A)	双曲正弦
	COSH(A)	双曲余弦
	TANH(A)	双曲正切
	COTH(A)	双曲余切
	SECH(A)	双曲正割
	CSCH(A)	双曲余割
	ASINH(A)	反双曲正弦
	ACOSH(A)	反双曲余弦
	ATANH(A)	反双曲正切
	ACOTH(A)	反双曲余切
	ASECH(A)	反双曲正割
	ACSCH(A)	反双曲余割

表 6.2　FNTLIB.TRC/FNTDLIB.TRC(三角函数库表)

包含文件	函　数	说　明
FNT.TRU/FNTD.TRU	COT(A)	余切
	SEC(A)	正割
	CSC(A)	余割
	ASIN(A)	反正弦
	ACOS(A)	反余弦
	ACOT(A)	反余切
	ASEC(A)	反正割
	ACSC(A)	反余割

表 6.3　FNMLIB.TRC(数学函数库表)

包含文件	函　数	说　明
FNM.TRU	LOGBASE(X,B)	以 B 为底 X 的对数
	ERF(X)	误差函数:2＊NORMAL(SGR(2)＊X)
	NORMAL(A,B)	正态分布 A 到 B 的概率
	NORMAL(X)	正态分布 O 到 X 的概率
	FAOTRL(N)	N 的阶乘
	BINOM(N,J)	二项式的系数
	BINOMPR(N,J,P)	二项式分布
	POISSON(M,J)	泊松分布

此时,编辑窗中源程序光标后插入包含文件,主要内容是:

LIBRARY“库文件名”

DECLARE DEF <函数名>[,函数名]

为保证先声明后调用的原则，往往把指定库的包含文件插在程序首部。

例 6.31　画反余切曲线 ACOT(X)，X 从－10 到＋10 取值。该程序需调用 FNTLIB.TRC 库文件中的 ACOT 函数，所以程序首部加入了该库的包含文件 FNT. TRU 的两个语句。又因要使用图形子程序，还指明使用 GRAPHLIB.TRU 库文件(请参见图形子程序库)。

```
! LIBGF PROG
LIBRARY "FNTLIB.TRC"
DECLARE DEF COT, SEC,CSC,ASIN,ACOS
DECLARE DEF ACOT,ASEC,ACSC
LIBRARY "GRAPHLIB.TRU"
OPEN #1;SCREEN.2,.8,.11,.9
SET WINDOW -10,10,0,PI
CALL FRAME
CALL TICKS(1,PI/4)
DIM TEMP(0 TO 64,2)
ASK WINDOW X1,X2,Y1,Y2
LET DELTA=(X2-X1)/64
FOR I=0 TO 64
    LET X=X1+I*DELTA
    LET Y=ACOT(X)
    IF Y<Y1 THEN LET Y=Y1
    IF Y>Y2 THEN LET Y=Y2
    LET TEMP(I,1)=X
    LET TEMP(I,2)=Y
NEXT I
FOR I=0 TO 64
    PLOT TEMP(I,1),TEMP(I,2);
NEXT I
PLOT
END
```

执行结果见图 6.4。

(2)图形子程序库

图形库文件 GRAPHLIB.TRU 包含 7 个图形子程序，它以 True BASIC 源程序形式(ASCII 码)存储在系统盘上，供用户调用，用户亦可以根据需要扩充该库文件。

库文件中包含的子程序名称，形式参数及功能说明如下：

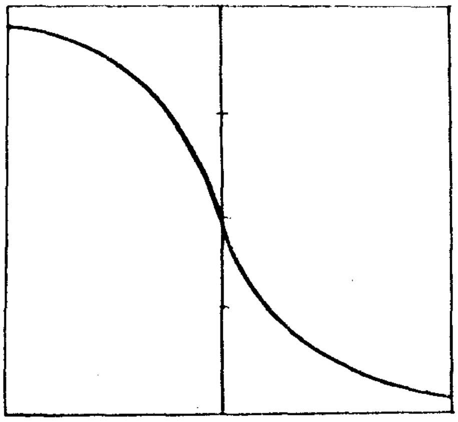


图 6.4　反余切曲线

①画圆弧子程序：ARC (X,Y,R,A1,A2)

在当前窗口中以坐标(X,Y)点为圆心，以 R 为半径，从角度 A1 到 A2 画圆弧，其中 A1，A2 采用度为单位。

②画坐标轴子程序：AXEC

在当前窗口上画 X 轴和 Y 轴。

③画坐标刻度线子程序：TICKS(a,b)

在当前窗口上画 X 和 Y 坐标轴，并在 X 坐标轴上以 a 为刻度单位画刻线，在 Y 轴上以 b 为刻度单位画刻线。

④画窗口边框子程序：FRAME

在当前窗口周围画一个边框，它利用 ASK WINDOW 语句自动获取窗口坐标。

⑤画函数曲线子程序：FPLOT(a,b)

在当前窗口上，X 从 a 到 b 区间变化，画出函数 F(X)的曲线，其中 F(X)是用户定义的外部函数。

⑥画多边形子程序：POLYGON(X1,X2,Y1,Y2,N)

在当前窗口上由坐标 X1,X2,Y1,Y2 所定义的框内画一个 N 边形。

⑦画棒形统计图子程序:BARS (data,n)

根据一维数组 data 中的数据画一张条形图,条形图窗口由数据自动确定。

图形库文件的使用:在用户程序调用图形库文件中的图形子程序时,首先在程序首部用如下语句:

LIBRARY GRAPHLIK. TRU

调入库文件,此后,程序用语句

CALL 子程序名<实参表>

调用指定的子程序,完成该子程序的作图功能。

例 6. 32　利用 GRAPHLIB. TRU 库文件中的子程序 FRAME,AXES,TICKS,FPLOT 画函数 F(X)=1/X 曲线。

```
! plot 4 prog
set mode "hires"
library "graphlib. tru"
open #1: screen. 2,. 8,. 1,. 9
set window 0,2,0,10
call frame
call ticks(. 5,1)
call fplot(0. 1,2)
end
def f(x)=1/x
```

执行结果见图 6. 5。

说明:

①图形库 GRAPHLIB. TRU 所有的作图功能皆用子程序表达,因此用 CALL 语句调用,不能用子图调用语句 DRAW。

②TICKS(0. 5,1)子程序画坐标轴和坐标刻度线,例中 X 轴以 0. 5 作单位,Y 轴以 1 作单位,X 轴画 5 条刻度线,Y 轴画 10 条刻度线。在画刻度线时,刻度线长短较难处理,如果与窗口无关,采用固定长度刻度线,显然不合理。例如用 0. 1 作 X 轴上刻度线高度,当 Y 轴坐标范围从 0 到 10 时,比较合适。但如果 Y 从 0 到 100 的范围时,刻度线仅为屏幕上 Y 长度的 1/1000,根本看不见。如果 Y 从 0 到 2 的范围,刻度线又显得太长了。子程序自动根据窗口大小和窗口占用的屏幕大小,按排并画出比较合理长短的刻度线。

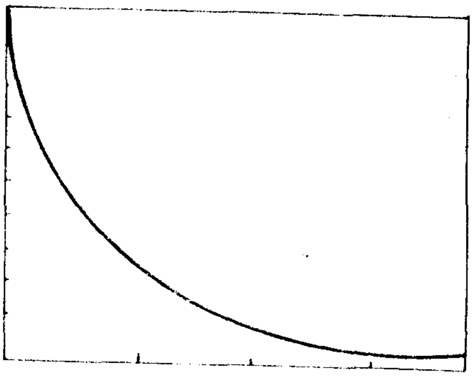


图 6. 5　函数 F(X)=1/X 曲线

③FPLOT 子程序画出程序外部定义函数曲线,子程序采用了出错处理结构,如外部定义函数 F(X)=1/X,在 X 等于 0 时,将出现被零除溢出错误而停机,保护措施使之仍正常工作,作相应处理。另外,子程序的直线插值自动采用 64 分段,保证函数曲线光滑,和不丢失因舍入误差最后一点。

(3)菜单库文件

菜单库文件 MENULIB. TRU 供用户设计菜单用,菜单是模块化程序设计中经常运用的一项技术。菜单库文件中包括 5 个子程序:

MENU_SET——打开窗口 #9,并设置其它参数;

MENU_ALL——显示菜单和提示,接受回答,显示回答,清除菜单窗口,返回到工作窗

口；

MENU——显示菜单且得到一个回答；

MENU_SHOW——显示菜单；

MENU_ASK——得到一个回答。

应用菜单库文件设计菜单的程序应在程序开始处加一条语句：

LIBRARY "MENULIB"

然后调用 MENU_SET 子程序，用它来打开菜单窗口#9，以及设置菜单的位置，颜色，项数，项的大小等参数，在调用 MENU_SET 以后，可使用其它子程序。

子程序的形式参数及功能：

①MENU_SET（WHERE$，C$，MAXENT，MAXLEN，MENU$，#9）

子程序打开#9号窗口，指定在#9号窗口中显示菜单。其中参数 WHERE$ 指定菜单在屏幕上的位置，可以在屏幕上/下部（“TOP”/“BOTTOM”），或在屏幕的左/右侧（“LEFT”/“RIGHT”）。在实在参数指定是“TOP”/“BOTTOM”时，则#9号窗口在屏幕的上/下部占用2行或3行，如果指定是“LEFT”/“RIGHT”，#9号窗口位于屏幕左/右侧的垂直窗口，宽为13个字符，长为22行。窗口位置指定字符串字母大小写等效。

参数 C$ 是设置菜单前景和背景的彩色字符串，例如实参指定“RED/BLUE”表示菜单在兰色背景上书写红色的字。

参数 MAXENT 指定菜单的项数，该数值不能大于10。

参数 MAXLEN 指定菜单中最长的项的字符个数，最大长度亦不能大于10个字符。参数 MANU$ 是一个压缩变量，它是 MENU_SET 子程序结果传递参数，供其它子程序使用。

参数#9亦是出口参数，它在该子程序中打开，由其它子程序使用。

②MENU_ALL(A$()，M，PROMPT$，ANS，MENU$，#N，#9)

参数 A$()是一个一维数组，数组元素是菜单中各项的内容。

参数 M 是 A$()一维数组的元素个数，也就是菜单实际项数，它不能大于 MENU_SET 子程序指定的 MAXENT 参数值，如果它小于 MAXENT 的值，则说明菜单中留有待扩充的项。

参数 PROMPT$ 是指定菜单使用时的选择提示字符串，它在工作窗口#N中显示。

参数 ANS 是出口参数，它给菜单使用程序传递用户选择第几项这一结果，它是一个1到M之间的数。

参数 MENU$ 是由 MENU_SET 产生的压缩串，不必改变它的内容。

参数#N是一个程序打开的工作窗口，N可以是0～999的常数。

参数#9是由 MENU_SET 打开的菜单显示窗口。

MENU_ALL 子程序完成在#9号窗口显示菜单，在#N工作窗口显示菜单选择提示。在用户选择回答后（菜单选择使用功能键），在工作窗口上显示用户回答，并检查回答的合法性，如果合法则清除#9窗，返回工作窗，如果不合法则在菜单窗口中显示出错信息。

③MENU(A$()，M，ANS，MENU$，#9)

子程序 MENU 的功能是显示菜单，并获得用户对菜单选择的回答。子程序的内容是以下两个子程序 MENU_SHOW 和 MENU_ASK。子程序 MENU 被 MENU_ALL 调用，MENU 子程序的参数意义和 MENU_ALL 相同。

④MENU_SHOW(A $ (),M,MENU $,#9)

子程序参数含义同 MENU_ALL,它的功能是显示菜单。

⑤MENU_ASK(M,ANS,MENU $,#9)

子程序参数同 MENU_ALL,功能是获得用户菜单选择的回答,供程序用。

例 6.33　利用菜单库文件设计菜单。

```
! menushow program
library "menulib"
dim weather $ (6)
mat read weather $
data hot, cold,rain,snow,sleet,stop
let where $ ="bottom"
let col $ ="red/blue"
let maxent=6
let maxlen=9
let m=6
open #1:screen.3,.7,.3,.7
call menu_set(where $ ,col $ ,maxent,maxlen,
    menu $ ,#9)
do
    call menu—all(weather $ ,m,"weather",
ans,menu $ ,#1,#9)
  select case ans
  case 1
    print "go swimming"
  case 2
    print "wear mittens!"
  case 3
    print "walk your pet duck!"
  case 4
    print "go skiing!"
  case 5
    print "read a book!"
  case else
      stop
  end select
loop
end
```

6.5　出错处理与出错函数

6.5.1　出错处理程序

在运行一个 True BASIC 程序时,可能由于某些原因而发生错误,例如试图对负数求平方根、被零除溢出错误、试图打开某个不存在的文件或文件操作错误等。它将停止程序的运行,并显示出错信息。因此,程序运行时出错,几乎都是“致命性”的。True BASIC 提供的出错处理结构,允许在用户程序中设置特定的“错误处理模块”,当程序运行时出错时,计算机不是停止程序运行,而是按“错误处理模块”的要求进行处理,抑制运行时出错,继续程序运行。

出错处理结构的格式:

```
WHEN ERROR IN                 ┐
…………………………………………          │
…………………………………………          ├保护模块
…………………………………………          ┘
USE                           ┐
…………………………………………          │
…………………………………………          ├错误处理模块
…………………………………………          ┘
```

END WHEN

功能：定义一个程序运行时出错处理结构。

说明：在 WHEN 语句与 USE 语句之间的语句块称为“保护模块”，这个语句块是用户要进行某种运算或处理可能出现运行时出错的程序段。

USE 语句与 END WHEN 语句之间的语句块称为“错误处理模块”，这个模块中有若干个错误处理语句，可根据需要设置。在正常情况下，程序执行 WHEN 与 USE 之间的“保护模块”，在此模块中如无错误发生时，则跳过“错误处理模块”，执行 END WHEN 后的语句；如果在“保护模块”执行过程中发生运行错误，则执行 USE 后的“错误处理模块”，若在″错误处理模块″中不再发生错误，则继续执行 END WHEN 后的语句。

在程序设计中在可能发生运行错误的地方加入出错处理结构，可以避免出错后的中止程序运行现象，这种方法对设计一个较大的程序时，常常是必须考虑的。

除函数、子程序或子图的定义模块外，保护模块中可以包含任何类型的语句。保护模块中允许有多层出错处理结构嵌套。但错误处理模块中不允许有另外的保护模块。

如果在保护模块中调用了一个函数或子程序，则它们的所有语句均被同一个错误处理模块所保护。而这些函数或子程序也可能有自已的错误处理程序，但它们是不管调用者是否有错误处理程序，在这种情况下，调用者的错误处理是函数处理的外围，即可能有几层处理。

可以使用 EXIT HANDLER 语句从嵌套的错误处理程序中退出，而避开某一层的处理。

在 WHEN ERROR IN 语句中，除可使用 ERROR 保留字外，为了与美国国家标准 BASIC 兼容，可以使用关键字 EXCEPTION。

6.5.2 **出错函数**

True BASIC 提供了几个出错函数以供用户在错误处理时使用，它们可以告诉用户错误的代码或错误信息的文字说明，或者是错误出现在主程序或子程序中以及所在的行号。

(1)EXTYPE 函数

EXTYPE 函数可指出当前所发生错误的代码(出错号)，它可以放在程序中的任何地方，但通常用于出错处理程序中。出错号的含义规定如下：

0	无错误发生
1～999	用户选用
1000 以上	True BASIC 提供的出错号

所有的 True BASIC 的出错信息及相应的出错号见有关附录。

例 6.34　应用 EXTYPE 函数的错误处理程序。

```
! demonstrat extype
DO
  WHEN error in
    INPUT prompt "Enter two numbers:";x,y
    let z=x/y
    PRINT "x=";x,"y=";y,"z=";z,
    PRINT "extype=";extype
  USE
    PRINT "sorry.","extype=";extype
    PRINT "please try again."
  END when
LOOP
END
...................................
ok. run
Enter two numbers:20,5
x=20     y=5     z=4     extype=0
Enter two numbers:5,0
```

```
sorry.          extype=3001
please try again.
Enter two numbers:
```

```
ctrl-break
ok.
```

通过这个例子可以理解出错处理程序的结构。当输入数据正确时(如 X=20,Y=5),仅执行保护模块,而跳过出错处理模块。但当输入数据为 X=5,Y=0 时,保护模块运行时发生了一个数被零除的错误,由于有了错误处理模块,该程序没有被停止运行,输出出错号后,请求重新输入数据,重新输入数据后程序可继续执行。运行结果中 EXTYPE 函数值为 0 表示无错误,EXTYPE 函数值为 3001 表示被零除的错误代码。这个程序中的 DO-LOOP 循环因没有带 WHILE 和 UNTIL 条件短语,所以是死循环,要终止这个循环可同时按 CTRL 和 BREAK 键。

(2)EXTEXT $ 函数

EXTEXT $ 函数给出当前发生错误的错误提示信息,如果没有出错,则 EXTEXT $ 函数返回空串,此函数可用于程序中的任何地方。

例 6.35　应用 EXTEXT $ 函数的错误处理程序,防止无用数据输入,停止程序运行。

```
! demonstrate extext $
PRINT "Enter a number(>=0):";
DO
    INPUT n
    WHEN error in
          CALL sqrt(n,t)
          EXIT DO
    USE
          PRINT extype;"——";extext $
          PRINT "please enter again."
    END WHEN
LOOP
PRINT "square root(";n;")=";t
END
SUB sqrt(n,t)
    LET t=sqr (n)
END SUB
```

```
…………………………………

ok. run
Enter a number(>=0):? 79
square root (79)=8.88819
ok. run
Enter a number(>=0):? -4
3005——SQR of negative number
please enter again.
? 86
square root (86)=9.27362
ok.
```

(3)EXLINE $ 函数

EXLINE $ 函数返回一个字符串,此串表示最近发生的错误在程序中的位置,如果无错误发生,它就返回空串,否则就列出出错的语句行号和错误所在的函数名或子程序名。形式如下:

LINE IN ROUTINE

其中 LINE 为出错的实际行号,ROUTINE 为错误发生于其中的子程序名或函数名。如果你的程序没有行号,True BASIC 将程序的第一行认为 1,第二行认为 2,等等,一直到程序结束。主程序的程序名为“MAIN”。

如果错误出现在某个子程序中,True BASIC 先给出有错的行一个逗号,然后再给出调用者的行号。如果调用者有多个层次,那么按照嵌套的相反顺序列表。但仅列出第一层及最后三层的调用者,其它中间调用层均被忽略。

例 6.36　EXLINE $ 函数的使用。

```
! demonstrate exline $
def badfunc(n)=1/n
sub badsub
   input prompt "Enter the value of n";n
   let x=badfunc(n)
   print "x=";x
end sub
when error in
   call badsub
use
   print "exline $ =";exline $
end when
end
...........................
ok. run
Enter the value of n 5
x=.2
ok. run
Enter the value of n 0
exline $ =2 in badfunc in Main,5 in badsub in
     Main,9 in Main
ok.
```

第一次运行时输入数据为 5，没有错误发生。第二次运行时输入数据为 0，发生了错误，由 EXLINE $ 函数返回错误在程序中的位置。错误在主程序中的自定义函数(BADFUNC)第 2 行，主程序中的子程序(BADSUB)的第 5 行，还有在主程序的第 9 行，其中行号 2，5，9 均是程序中的自然行号。

当调用函数或子程序时(调用有多个层次时)发生错误，则列出错误的顺序与调用的顺序刚好相反。

(4)EXIT HANDLER 语句

EXIT HANDLER 语句仅用于出错处理程序中。当某一出错处理模块不能恰当地处理某一错误时，则可安排 EXIT HANDLER 语句，交给外层的出错结构处理，这样就扩大了排除错误的范围。如果在外层再没有别的出错处理程序，就有 True BASIC 来处理它，并显示适当的出错信息。

(5)CAUSE ERROR 语句

CAUSE ERROR 语句用来定义用户的出错代码和错误提示信息，其格式为：

CAUSE ERROR＜数值表达式＞，＜字符串表达式＞

其中＜数值表达式＞表示出错代码，＜字符串表达式＞表示错误提示信息，错误提示信息用双引号括上。

例 6.37　EXIT HANDLER 语句的使用。

```
! demonstrate cause error
INPUT prompt "a,b,c=";a,b,c
WHEN error in
    LET result=a * b/c
USE
    IF extype<>1000 and extype<>3001 then
 EXIT HANDLER
    END IF
    LET result=maxnum
END WHEN
    PRINT "result=";result
END
..............................
ok. run
a,b,c=10,2,4
result=5
ok. run
a,b,c=10,2,0
result=3.59539e+308
ok.
```

例中出错代码 1000 表示有溢出错误，出错代码 3001 表示被零除错误。在这两种之一错误发生时，出错处理模块用 MAXNUM 给 RESULT 赋值来排除错误。如果其它错误无法排

除，则用 EXIT HANDLER 语句，跳出该处理模块，由 True BASIC 处理，显示出错信息。

(6)出错处理示例

例 6.38　编程求解一元二次方程 $ax^2+bx+c=0$。

在输入系数时，如果 a=0 或 a=0，b=0，都将出现致命性错误。本例使用出错处理结构嵌套，使输入无用数据时不中止程序执行，要求重输入数据，直到输入正确，输出正确结果。

```
! solution of a quadratic equation
DO
    LET fn=0
    INPUT PROMPT"a,b,c=";a,b,c
    WHEN ERROR IN
      LET r=-b/(2*a)
    USE
      LET fn=extype
      PRINT "a=0";
      WHEN ERROR IN
        LET x=-c/b
        PRINT "x=";x
      USE
        LET fn=EXTYPE
        PRINT "b=0";
        PRINT "no result!"
      END WHEN
    END WHEN
    IF fn=3001 THEN
      PRINT "re enter"
    ELSE
      EXIT DO
    END IF
LOOP
LET d=b*b-4*a*c
SELECT CASE d
      CASE 0
      PRINT "x1=x2=";r
CASE IS>0
      PRINT "x1="; r+sqr(d)/(2*a),
      PRINT "x2="; r-sqr(d)/(2*a)
CASE ELSE
      PRINT "x1=";r;"+";
      PRINT sqr(-d)/(2*a);"i",
      PRINT "x2=";r;"-";
      PRINT sqr(-d)/(2*a);"i"
END SELECT
END
....................................
ok. run
a,b,c=6,4,3
x1=-.333333+.62361i
x2=-.333333-.62361i
ok. run
a,b,c=0,5,20
a=0 x=-4
re enter
a,b,c=2,0,5
x1=0+1.58114i        x2=0-1.58114i
ok.
```

小　　结

True BASIC 除提供了许多标准函数外，还允许用户自己定义函数和子程序，建立用户自己的程序库。函数和子程序都是功能语句模块，这给结构化、模块化程序设计提供了有力的工具。

函数和子程序都有内部和外部之分。凡在调用程序中定义的函数或子程序，称内部函数或内部子程序，它们和调用程序是一个程序单位，共享变量，无论是调用程序、函数或子程序中的变量都在整个程序单位内有效，称为全程变量；凡在调用程序外部，包括在程序库文件中定义的函数或子程序称外部函数或外部子程序，它们都是独立的程序单位，其中变量仅限

于自身所在的那个程序单位中使用，故称为局部变量。

函数或子程序定义中的参数，称为形式参数，而在函数或子程序的调用中使用的参数称为实在参数.形式参数是局部变量，只在定义的范围内有效。根据形式参数的功能有值参和变量参数之分，值参数用来接收调用程序传递要求处理的数据、数组、文件或图形窗口，变量参数用来发送该模块处理的结果数据。函数定义中只有值参数，同时不允许使用通道参数。

自定义函数有单行自定义函数和多行自定义函数。函数定义可以带形式参数，也可不带形式参数。形式参数是函数的自变量，调用时实在参数的值赋于形式参数，由函数名返回调用结果。根据函数返回值的类型，函数又有数值型和字符串型函数之分，字符串型函数名后必须有“＄”符号。

子程序可带参数，亦可不带参数。内部子程序可以通过全程变量或参数返回子程序的调用结果，外部子程序只能通过参数返回调用结果。调用时不开辟存贮单元，只利用相应的实参所占的内存单元而建立地址联系，子程序调用结果按建立的地址联系写入实参所占的内存单元，调用结束，撤消地址联系。

由于外部函数和外部子程序中所使用的变量为局部变量，每次调用时，所有变量都要初始化(数变量为 0，串变量为空串)，且不会改变调用程序中变量的值(即使是同名)，所以外部函数和外部子程序可以递归调用。

True BASIC 允许把若干个外部函数和外部子程序汇集一起构成一个程序库，以库文件形式存入磁盘，需要时可供其它程序单位调用，共享程序库资源。

习　题

1. 执行下列各程序后，输出的结果是什么？

(1)
```
LET X$="TR"
LET Y$="UE"
LET Z$=X$&Y$&"100"
LET A$=Z$[3:8]
PRINT A$,A$&"10"
END
```

(2)
```
LET X=4
LET Y$=CHR$(30+X)&"K"
PRINT ORD(Y$[1:1])+ORD(Y$[2:2])
END
```

(3)
```
LET T$="ABMNP"
LET S$="CD"
LET X=LEN(T$&S$&"") * LEN(T$)
LET Y$=STR$(X)
PRINT Y$[1:2]&"4"
END
```

2. 有一长度不大于 40 的字符串，已知其中包含有两个字符“E”，编程求处于这两个字符"E"中间的字符个数。

3. 从键盘输入一个含有若干个“X”的字符串，试编程统计其中含有“X”的个数。

4. 编写一个程序，能把从键盘输入的两位十六进制数转换为十进制数。

5. 有一长度大于 2 的字符串，现要求分别舍去这个字符串的左右两端的一个字符，并将小写字母改成大写字母，然后显示输出。

6. 用重复字符串函数打印一行“—”号。

7. 用自定义函数的方法求下列各式：

(1)$Y=(1-A)/(1+A+A^2)[(1-B)(1+B+B^2)+(1-C)/(1+C+C^2)]$

(2)$Z=[(\sin X^2+\sin Y^2+K)/\sqrt{\sin X^2+\sin Y^2}]\cdot \mathrm{tg}^2(\sin X^2+\sin Y^2)$

(3)$T=(F(X)+F(Y^2))/((F(X+1)+Y^2)\cdot F(F(Z))$

其中 $F(U)=U-U^2/6+U^5/120$

8. 编写一个程序，求半径分别为 1,2,3,4,5 的圆面积。

9. 当 $X=3$ 时分别求下列各式的值：

(1) $F_1=4X^2+\sqrt{X+1}$

(2) $F_2=4X^2+\sqrt{X+1}+10$

(3) $F_3=2(2X^2+\sqrt{X+1}(4X^2+\sqrt{X+1}+10))$

10. 定义函数 LENGTH $ ，它以一个字符串为变量，如果该字符串不多于三个字符则函数值为“SHORT”，如多于 7 个字符则为“LONG”，其它情况为“MEDIUM”。

11. 计算 N 维空间中点 $X(X_1,X_2,\cdots,X_N)$ 和 $Y(Y_1,Y_2,\cdots,Y_N)$ 之间的距离：

$$R(X,Y)=\sqrt{\sum_{i=1}^{N}(X_i-Y_i)^2}$$

12. 试用多行自定义函数定义编制程序，构成如下几个标准函数的程序清单：

(1) 返回自变量 X 的符号：SGN(X)

(2) 返回 X，Y 中的最大值和最小值：MAX(X,Y)，MIN(X,Y)

(3) 返回 X 的正弦值：SIN(X)

提示：按幂级数公式计算：

$$\sin(X)=X/1!-X^3/3!+X^5/5!-X^7/7!+\cdots-X^{4N-1}/(4N-1)!+X^{4N+1}/(4N+1)!$$

并且当最后一项的绝对值 $<10^{-5}$ 时为终止。

13. 指出下列程序中的全程变量和局部变量，写出它们的运行结果：

```
(1) DEF F(X)
       IF X * Y<0 THEN
         LET F=X * Y/2
       ELSE
         LET F=SQR(X * Y)
       END IF
    END DEF
    DO
      READ A,Y
      PRINT F(A)
    LOOP UNTIL A=0
    DATA 5,-3,8,2,-4,-9,0,7
    END
(2)FOR I=1 TO 10
     CALL SUBR
     PRINT I
   NEXT I
   END
   SUB SUBR
     LET I=I+1
     PRINT I
   END  SUB
```

14. 如有两个正整数 A、B 的最大公约数为 D，则可得 A、B 的最小公倍数 M=AB/D。利用这个结论编出求两个数最小公倍数的子程序，并求四个数 456，133，76，252 的最小公倍数。

15. 求出三组随机数，每组三个数，用子程序输出这三组数，用子程序找出每组中最大的数。

16. 试将下列单词按英文字母顺序排列出来：BEAD，ABOUT，CAPACITER，DAMP，FAIL，BEEP，IMPEDANCE，MEET，OIL，POWER，SETTING，DAKER。

17. 有自然数 1，2，3，…，N（N<100），除 1 和 2 之外其余的自然数之和是 100 的整数倍，试求 N 的值。（提示：$3+4+5+\cdots+N=(N-2)(N-3)/2$，当 $(N-2)(N-3)/2$ 的值为 100 的整数倍时 N 就是所求的值）。

18. 将 N 个数中的负数选出来，使用子程序将它们置入一维数组，再用一个子程序将它们排序（从大到小递降）并输出。

19. 产生一个 10×10 的二位随机整数方阵，试编程完成下列任务：先求出每一行的最大值和每一列的最小值，然后求 10 个最小值中的最小值，10 个最小值中的最大值。

20. 用递归子程序求 N 个正整数的最小公倍数。

第7章　文　　件

BASIC 语言不仅可以解决各类数值计算问题，而且广泛地应用于各种非数值处理的事务管理。在计算机用于复杂的科学计算和事务管理（如职工档案管理、图书资料、情报管理、银行开户存取管理、调查采集数据的统计分析等）时，往往需要处理大量的各类数据。BASIC 语言提供的三个数据输入语句：LET，READ/DATA 语句必须在程序中提供数据，由于数据和程序是同一个程序单位，数据只限所在程序使用，不可能为程序共享，修改不方便，同时为确保程序运行，数据量不能过大。INPUT 语句解决了程序与数据分离的问题，增强了程序的灵活性，但键盘对话式的数据输入方法不仅速度慢，一旦出错便前功尽弃，大量数据每次重复输入浪费时间、精力，数据共享也不可能。由于计算机的内存容量有限，特别是微机，同时内存不能长期保存信息，因此，在处理大量数据时，必须解决信息的长期存储和数据资源共享问题，文件便由此产生。

文件（FILES）是存储在外存储设备（磁盘）上的信息集合。按存储的信息是程序或数据，分为程序文件和数据文件两类。两类文件解决了程序、数据分离以及大量数据和程序的长期存储问题。文件一旦建立便可反复使用，从而提高了工作效率；数据文件可以传递给不同程序共享，提高了数据资源的利用率。

本章简单介绍 True BASIC 程序文件的编辑、存储和使用，介绍数据文件的建立、读写操作和应用。

7.1　MS—DOS 文件

文件由计算机操作系统管理。MS—DOS 统一管理 True BASIC 的文件，提供包括驱动器、树结构目录等项的文件命名格式。

（1）文件命名和文件的类别

无论是程序文件或是数据文件，为了加以区别，每个文件都有一个标识符，称文件引用名。按 MS—DOS 规定，文件引用名由四部分组成：

[<盘符：>][<目录路径>]<文件名>[<.扩展名>]

其中，盘符可用 A 或 B 表示软盘驱动器，C 或 D 表示硬盘驱动器，指定文件存储盘所在的驱动器。如果文件就在当前驱动器的盘上，则盘符可以省略，文件在当前目录中，目录路径可以省略。

文件名最长允许有 8 个字符，至少 1 个字符。可用作文件命名的字符是英文字母、数字和除空格符、*、?、\、.、/、,、;（DOS 另有作用）外的符号。

扩展名用作指定文件的类别，最多由 3 个字符组成。在某些特定情况下，扩展名可以省略，有些扩展名 DOS 有约定。例如：

COM：可执行的二进制代码文件

EXE：可执行的浮动代码文件

BAT：可执行的批处理文件

LIB：库文件

HLP：自助文件

BAK：备份文件

TRU：True BASIC 源程序文件

TRC：True BASIC 编译程序文件

采用小写字母的文件名将自动地转为大写。在 True BASIC 中，源程序存储时如果省略扩展名，则系统自动加上“.TRU ”，对编译文件则加上“.TRC ”。以下是合法的文件名：

TSORT.TRU ，PROG1 ，1993.JAN ，ABC.DAT ，Text.TRC ，TUDGET.%

(2) 保留文件名

DOS 操作系统将一些外部设备视为特殊的文件，并有专用的文件名，称保留文件名，它们为系统使用，不能用作一般文件命名。保留的文件名有：

CON：	控制台(键盘/显示器)
PRN.或 LPT1.	第一个并行打印机
LPT2.	第二个并行打印机
COM1.	第一个异步通讯适配器接口
COM2.	第二个异步通讯适配器接口
NUL.	虚拟设备

7.2 有关文件操作的系统命令

程序文件保存着 True BASIC 的源程序或编译程序，分别称源程序文件和编译文件。程序文件由 True BASIC 系统命令管理，允许装入内存并运行程序。在系统中，ASCII 编码行结构的字符集称文本或正文，相应的存储文件称文本文件或正文文件。源程序文件是正文文件，允许装入内存，在编辑窗中显示、修改、再存储，编译文件不能显示、修改。

(1) 当前程序存盘命令

格式：SAVE <文件引用名>

它命令将当前在内存中的程序写到指定的磁盘上，建立一个由文件引用名命名的新的程序文件。执行该命令时，系统将要证实没有与引用文件名同名文件后，再写到指定盘上，否则出错。如果文件引用名中的扩展名省略，则系统自动判断，如内存中是源程序，文件名后加“.TRU”，若是编译程序，则加上“.TRC”的扩展名。

(2) 同名文件替换命令

格式：REPLACE [<文件引用名>]

它命令将当前程序存盘替代同名文件。该命令的文件引用名可以省略，表示它替代存储的就是调入的程序文件，被替代的文件不再存在。该命令用于源程序文件或正文文件，经编辑修改后再存储。

(3) 文件装入命令

格式：OLD <文件引用名>

它命令把磁盘上的指定文件装入内存，同时编辑窗清屏；若调入的是源程序，则显示程序清

单，编译程序不可显示。使用该命令时，指定的文件必须存在，否则出错。注意，在使用 OLD 命令前，保存原内存中的源程序，否则可能丢失。

（4）源程序编译命令

格式：COMPILE

它命令编译内存中存储的源程序。编译完成后屏幕上显示的源程序被清除，内存中亦被编译程序替代，此时用 SAVE 命令可建立一个编译文件。为避免源程序丢失，编译前应保存源程序。编译程序占用存储空间少、运行速度快，由于它不可显示，程序有保密性，所以对一些成熟、常用的程序，包括外部函数、外部子程序汇集共享的程序库都加编译，以编译文件保存。

（5）文件目录显示命令

格式：FILES [<文件名引用>]

它命令显示文件目录。文件引用名缺省时，显示当前驱动器盘上扩展名为.TRU 的文件目录。该命令与 DOS 目录显示命令类似，可以使用文件名多义符显示某类文件的目录。例如 FILES *.* 或 FILES B:*.* 可显示当前驱动器或 B 盘上的全部文件目录。

（6）文件删除命令

格式：UNSAVE <文件引用名>

它命令从磁盘上删除文件引用名指定的文件。文件扩展名缺省时，在多个同名文件中只删除扩展名是 .TRU 的文件。无论程序文件、数据文件都可用该命令删除，事先不做任何提示，使用时要谨慎。该命令的文件名不能缺省，并且指定的文件必定存在，否则出错。

（7）其它命令

NEW——命令清内存和编辑窗，建立一个无名、无内容的源程序或文本；

RUN——命令运行内存中的程序；

LIST——命令打印输出内存中的源程序或正文文件，LIST BLOCK 打印当前文件由块标设定块的部分内容。编译程序不能打印输出。

系统命令均在背景窗的命令行下达，立即执行见效。本节介绍的系统命令中有关程序处理有 RUN，COMPILE；有关正文文件处理的命令有 OLD，SAVE，NEW，LIST REPLACE，UNSAVE，FILES。

注意，正文文件分为源程序文件和正文数据文件两类。源程序文件的内容是程序（BASIC 语句行的集合），它是可以编译、运行的正文文件；正文数据文件是字符行的集合，不能运行，但它受有关数据文件操作语句控制，作为数据由程序访问处理。由于两类文件在 True BASIC 中统称正文文件，因此在具体应用时应根据不同文件的性质分别处理。

7.3 数据文件

数据文件是存储在磁盘上相关数据的集合。数据文件一旦建立可以多次使用，长期保存，作为资源为多个程序共享。按数据存储方式分为固定格式文件和自由格式文件；按数据读、写的访问方式分为顺序文件和随机文件。顺序文件按顺序存储行格式数据或自由格式的字节流，顺序文件数据读出必须从头开始，数据写入时，只能在文件尾部添加。随机文件以记录作存取单位，每个记录都有相同的长度，记录编号，存储登录，因此可直接按记录号随机访问记录数据。

True BASIC 有三种数据文件：正文文件，记录文件，字节文件。正文文件是行格式的顺序文件，字节文件属自由格式的顺序文件，记录文件是记录格式的随机访问文件。

正文文件是字符行的集合，每行由 ASCII 编码的字符组成，回车符作行的结束标志。正文文件和源程序文件一样，可以直接用系统的文件处理命令建立，或将已存在的正文文件装入内存，在编辑窗中显示、编辑、打印、修改和再存储，不同的是正文数据文件不能运行。在程序控制正文文件读写时，与程序控制计算机控制台操作（键盘输入、屏幕输出）类似，正文文件中的数据也可象在屏幕显示一样安排行中的位置和格式。由于正文文件遵循 IBM—PC 的格式规定，可以适应在多种语言中建立和使用，用作不同语言间的数据交换。

记录文件是记录的集合。每个记录有固定的字节长度，一个记录可以只包含一个称字段的数据项。字段的数据类型可以是字符串或是数值；也可在一个记录中包含有多个类型相同或不相同的字段。每个记录有一个顺序号（自然数），标识某个特定记录，程序经记录号实施对某个特定记录的数据随机存取操作。在所有能用 True BASIC 语言的机器上，记录文件都具有相同的格式，可互相通用。

字节文件是由 ASCII 字节编码的字符流组成，数值转换为数字字符串，采用字节存储。由于字节文件给程序提供了逐个字符处理的能力，更适用于自由格式数据操作，使之可以操作任何结构的数据，同样可用作不同语言间的数据交换。

尽管 True BASIC 三种数据文件读、写操作语句各不相同，但程序使用过程都一样，分为打开文件、文件处理、关闭文件三个步骤。数据文件只有打开后才能进行数据读、写操作，否则出错。打开文件的作用是指定操作的文件，打开与文件相关联的数据传输通道、缓冲区、指定文件访问方式，指定或校核它是新建文件或是已经存在的文件、文件类型，对记录文件还可指定或校核记录长度。程序按文件打开的方式存取文件数据，处理数据。文件访问结束后，关闭通道，释放缓冲区，关闭文件，以免数据丢失。

(1) 通道

通道是数据传输的通路，是与打开文件相关联的一片内存数据缓冲区。True BASIC 最多允许打开 10 个通道，同时使用 10 个文件。通道由标识符“#”后跟代表通道号的数值表达式组成，通道号的取值范围是 1～999 的自然数。#0 是系统专用通道，不能用作数据文件通道，它代表计算机控制台（键盘与屏幕），总是打开着，不能被关闭。数据文件通道由 OPEN 语句打开，CLOSE 语句关闭。

① 局部通道和全程通道。外部函数或外部子程序是一个独立的程序单位，它们可以定义局部通道，正象可以定义局部变量一样。局部通道只局限于定义它的程序单位有效，它随程序单位调入运行而建立，程序单位运行结束自动彻消。全程通道在整个程序包括其中的内部函数和内部子程序皆有效，正象全程序变量一样，全程通道在程序单位中的模块间传递使用。全程通道使用时必须十分谨慎，以免通道冲突。

②通道参数。无论通道打开与否，都可以将通道作参数传递给子程序。子程序可以随意处理通道，使用数据文件，包括通道打开和关闭。形式通道参数亦具局部性。注意：函数不允许使用通道参数。子程序的形式通道参数只能用 1～999 的正整数，不允许使用变量或表达式。例如，#3，#99 是合法的通道形式参数，但 #N 则是非法参数。

(2) 打开数据文件 OPEN 语句

OPEN 语句用作打开数据文件及其相关联的通道，根据文件使用情况指定使用参数，进

行核对，保证文件正确被访问。OPEN 语句的格式：

OPEN #<通道>:NAME<文件引用名>[,ACCESS<访问方式>][,CREATE<文件方式>][,ORGANIZATION <文件类别>][,RECSIZE <记录长度>]

其中，NAME 和通道两项不可缺省，它指定要打开文件的引用名（所在盘、目录路径、文件名及扩展名）和与其相关联的通道。<文件引用名>是一个字符串表达式，<通道> 则是一个数值表达式，自动四舍五入取整，取值范围是 1～999 的正整数。

可选项 CREATE：用作核校新建文件或是已存在的文件。参数 NEW 指定<文件引用名>是要新建立的文件，如果该文件原来存在，则出错。OLD 参数指定的是一个已存在的文件，如果该文件不存在，则出错。参数 NEWOLD 比较灵活，如果文件不存在，则建立该文件，如果已经存在，则打开使用之。可选项省略时，系统取 OLD 参数。

可选项 ACCESS：指定核对打开文件的访问方式。参数 INPUT 指定为只读方式，在此方式下不允许写，亦不允许删除文件内容，新建立的文件不允许以 INPUT 方式打开。参数 OUTPUT 指定只写方式，在此方式下允许删除文件内容。参数 OUTIN 指定为读、写方式，它具有 INPUT、OUTPUT 两者的功能；可选项省略时，系统自动取 OUTIN 参数。

可选项 ORGANIZATION：校核所打开文件的类型，参数 TEXT，RECORD，BYTE 分别指定正文文件、记录文件、字节文件，文件类型不符则出错。新建立的文件用它指定文件的类型。如果未指定文件类型，则按程序中使用方式自动确定。使用已存在的文件时，用它核对打开文件的类型，类型不匹配则出错。如果缺省，则按已存在文件类型使用。

可选项 RECSIZE：只用于记录文件和字节文件读出操作，正文文件不用。对新建立的记录文件用它指定每个记录的字节数。对已存在的记录文件用它核对记录长度，若与原文件记录长度不符，则出错，此时亦可省略，表示按原文件记录长度操作。如果原文件内容已被全部删除，则用它重新定义记录长度。字节文件读出时用它指定每次读出的字节数。

OPEN 语句的可选项间用逗号分隔，排列顺序随意。可选项规定的参数可以直接使用，亦可以先赋给字符串变量，用字符串变量表示。

例 7.1

```
LET N$ ="B:RECFILE"
LET T$ ="RECORD"
OPEN #1:NAME N$,CREATE NEW,ORGANIZATION T$,RECSIZE 20
```

在 B 驱动器盘上建立称 RECFILE 的记录文件，记录长度为 20 个字节，使用 #1 通道，自动设定读写访问方式，并核对 B 盘上原来没有 RECFILE 同名文件，否则出错。

例 7.2　`OPEN #2:NAME "STD",ORGANIZATION TEXT`

打开当前驱动器盘上称 STD 的正文文件，使用 #2 通道，访问方式为读写方式。如果称 STD 的正文文件原来不存在，则建立该文件；若存在，核对是否为正文文件。是，则使用；否则出错。

例 7.3　`OPEN #3:NAME "TEACH"`

打开 TEACH 文件，使用 #3 通道，自动设定为 OUTIN 访问方式和 OLD 文件方式。若文件不存在，则出错；若文件存在，按原文件类型使用该文件。

(3) 文件关闭与删除语句

CLOSE 语句关闭文件，切断文件与通道的联系，释放缓冲区。语句格式为：

CLOSE #<通道>

无论通道是否已经打开均可用 CLOSE 语句关闭，关闭的通道再次关闭亦不会出错。

ERASE 语句删除一个已打开文件的内容，使文件成为有名无实的空文件。语句格式：

ERASE #<通道>

使用该语句一定要十分慎重，因执行该语句后，相应文件的内容将全部删除。它用于已存在文件删除内容后重写数据，或重新指定文件的类型或记录长度。内容删除后的空文件允许关闭或再打开。

(4) 文件参数设置语句

数据文件进行存取操作时，文件指针指出操作的当前位置，文件指针可用设置语句设置；对记录文件可用记录长度设置语句指定或校核记录长度；对正文文件可设置文件的行宽和域宽。表 7.1 列出各种文件参数设置语句及其作用。

表 7.1 文件参数设置语句

设置语句	作 用	适用文件类型
SET #<通道>：POINTER BEGIN 或 RESET #<通道>：BEGIN	指针指向文件开始	全部
SET #<通道>：POINTER END 或 RESET #<通道>：END	指针指向文件末尾	全部
SET #<通道>：POINTER NEXT 或 RESET #<通道>：NEXT	指针指向当前记录的下一个记录	记录文件
SET#<通道>：POINTER SAME 或 RESET #<通道>：SAME	记录访问后，指针指向保持不变	记录文件
SET #<通道>：RECORD <记录号> 或 RESET #<通道>：RECORD <记录号>	指针指向记录号指定有记录	记录文件
SET #<通道>：RECSIZE <数值表达式>	设置记录字节长度	记录文件；字节文件读出
SET #<通道>：MARGIN <数值表达式>	设置行宽字节数	正文文件
SET #<通道>：ZONEWIDTH<数值表达式>	设置域宽字节数	正文文件

说明：

① 文件打开后，指针指向文件开始 (BEGIN)；表中<记录号>，<通道>为数值表达式，其值分别代表数据文件当前记录号和相联的通道号。

② 正文文件、字节文件只能从头开始读，在文件末尾添加数据。在读写方式打开时，可用文件指针控制读、写操作。

③ 记录文件是随机访问文件，记录长度可以在 OPEN 语句的 RECSIZE 中设置，也可用 SET 语句设置。当打开的记录文件已经存在时，不允许再用 SET 语句设置记录长度，否则出错。在原记录文件内容全部删除后，只能用 SET 语句重新定义不同的记录长度。

④ 正文文件写操作是屏幕输出的映像，正文文件的行、域宽度用带通道的 SET 语句设置，同样在缺省时，行宽为 80 个字符，域宽为 16 个字符。

(5) 文件参数查询语句

凡可用 SET 语句或 OPEN 语句设置的文件参数，均可由 ASK 语句查询，获得设置值。表 7.2 列出文件各参数查询语句格式及说明。

表 7.2 文件查询语句

查词语句	说　明
ASK #<通道>;NAME <变量 $>	字符串变量返回文件名。图形窗口无名，为空串
ASK #<通道>;FILESIZE <变量>	数值变量返回文件大小，正文文件、字节文件字节总数，记录文件是记录数
ASK #<通道>;ACCESS <变量 $>	字符变量返回当前文件访问方式
ASK #<通道>;ORGANIZATION <变量 $>	字串变量返回当前文件类型。若返回"WINDOW"表明是图形窗口
ASK #<通道>;POINTER <变量 $>	字串变量表示文件指针当前位置，合法字串为"BEGIN"，"END"和"MIDDLE"，窗口指针总是"MIDDLE"
ASK #<通道>;RECORD <变量>	返回文件指针位置，记录文件是当前记录号，正文文件是行数，字节文件是字节数
ASK #<通道>;MARGIN <变量>	返回正文文件行宽字节数，其它文件为 0
ASK #<通道>;ZONEWIDTH <变量>	返回正文文件域宽字节数，其它文件为 0

说明：

① ASK 语句只能用来查询已打开文件的参数，否则出错。表中<变量 $>代表字符串变量，<变量>代表数值型变量，其中存储查询返回的结果。

② 图形的窗口参数查询，将在第 8 章中介绍。

(6) 文件测试逻辑函数

True BASIC 提供两个文件指针位置测试函数：

①MORE #<通道>，如果打开文件的指针指向文件末尾，为假值，否则为真值。

②END #<通道>，如果打开文件的指针指向文件末尾，为真值，否则为假值。

7.4 正文文件

正文文件是行格式顺序访问文件，它由一行一行字符组成，回车符是字符行的结束标志，行是正文文件的读、写操作单位。正文文件的读、写操作可认为是控制台操作的映像，读、写是带通道的 PRINT、INPUT 语句，与控制台读、写几乎相同。区别在于操作对象不同，读操作是从正文文件中读出数据，不是从键盘上读数据；数据输出亦不再是屏幕，而写入正文文件，尽管正文文件中数据存储安排十分类似于屏幕显示。此外，键盘输入允许出错再输入，数据不足再增加，但正文文件读出出错是程序致命性的错误。正文文件允许重读，添加数据，遵循的原则是：只能从文件开始读出，在文件末尾添加数据，只要不是以只读方式打开文件，可删除全部文件内容重写，没有数据修改操作功能。由于正文文件是 ASCII 编码文件，可用系

统命令 OLD 调入内存，在编辑窗显示，全屏幕编辑或用 LIST 命令打印输出，修改后的文件用 REPLACE 命令再存储，存储格式和行中的位置控制与屏幕上显示基本上一致。使正文文件建立、编辑和监示十分方便。

(1) 正文文件写入操作

正文文件除用系统命令建立、修改外，当正文文件以只写方式(OUTPUT)或读写方式(OUTIN)打开时，可由程序建立和写入数据。正文文件数据写入语句格式有下列三种：

PRINT #<通道>:[<表达式>][{,/;<表达式>}]

PRINT #<通道>,USING <格式串>:<输出项表>

MAT PRINT #<通道>,USING <格式串>:<输出项表>

数据写入正文文件的位置、格式和屏幕输出效果基本相同，以字符行为单位，PRINT 语句中的逗号和分号表示在同一行有多个数据项，按域格式或紧凑格式排列，数值型数据自动变换为数字字符串，每行以回车符结束。唯一不同的是屏幕一行只能显示 80 个字符，在列表长度超过 80 列时，自动换行，在正文文件中一行最长可达 32767 个字符。文件行宽可以设置，设置语句格式为：

SET #<通道>:MARGIN <数值表达式>

其中数值表达式的值应在 1～32767 间的正整数，缺省值是 80。文件域宽也可设置，设置语句格式为：

SET #<通道>:ZONEWIDTH <数值表达式>

其中数值表达式的值应在 1—行宽值间的正整数，代表域宽的字节数。注意，域宽最大亦不能超过行宽，否则出错。域宽缺省值是 16。

第二、第三种正文文件写入语句作用亦与屏幕输出类似，不再细叙。

例 7.4　在驱动器 B 盘上建立称 TXTFILE 的正文文件，存储学生成绩，每个学生有姓名、政治、英语、计算机成绩和三门课的平均成绩 5 项，以域格式存储，每域的宽度是 12 个字符。由键盘对话输入，其中平均成绩由程序计算填入。

```
   ! Create a text file called "txtfile"
LET n$ ="b:txtfile"
LET ch=1
OPEN #ch:NAME n$ ,ORGANIZATION text,CREATE newold
ERASE #ch
SET #ch:ZONEWIDTH 12
DO
    INPUT PROMPT "Name :":name$
    IF LCASE$ (name$ )="stop" THEN EXIT DO
    INPUT PROMPT "more data p,e,c :" :p,e,c
    PRINT #ch:name$ ,p,e,c,
    LET a=(p+e+c)/3
    PRINT #ch: a
LOOP
CLOSE #ch
END
```

说明：

① 文件打开语句中指定 B 盘上名为 TXTFILE 的文件是正文文件，建立方式是 NEWOLD，访问方式省略，系统指定为 OUTIN 方式。在打开文件后，不论原文件存在否，首先用删除语句 ERASE 删除文件内容，然后将键盘读入数据写入文件。因此该程序较灵活，既可用于新建 TXTFILE 文件，又可重新改写该文件，将建立视为空文件重写，在程序控制下，只有重写才能修改正文文件。如果文件建立方式采用 NEW，则程序只适用于 TXTFILE 文件初次建立使用，再次使用时会出现“FILE already exists” 的错误。文件虽然以读写方式打开，但本例只用写方式，ERASE 语句保证文件指针指向空文件的末尾(END)。

② 数据由键盘输入，在要求输入姓名时，如果键入 STOP，程序终止运行。在正文文件中同行数据的安排按域格式，域宽为 12 列。可在系统命令下，用 OLD 命令将正文文件调入内存、显示、修改或再存储，或用 LIST 命令打印 TXTFILE 正文文件。

例 7.5　对已建立的 TXTFILE 正文文件添加数据

```
! Append for a text file called "txtfile"
LET n$ = "b:txtfile"
LET ch=1
OPEN #ch:NAME n$
SET #ch:POINTER END
SET #ch:ZONEWIDTH 12
DO
   INPUT PROMPT "Name :":name$
     IF LCASE$ (name$)="stop" THEN EXIT DO
     INPUT PROMPT "more data:":p,e,c
     PRINT #ch:name$ ,p,e,c,
     LET a=(p+e+c)/3
     PRINT #ch:a
LOOP
CLOSE #ch
END
```

说明：该程序打开文件语句不用任何可选项，系统自动指定建立方式是 OLD，访问方式是 OUTIN，类型由打开文件决定是 TEXT。正文文件只能在文件尾部添加，因此文件打开后首先将指针指向文件末尾，其它情况和上例相同。

(2) 正文文件读出操作

正文文件以只读或读写方式打开时，语句 INPUT 读出数据，它与键盘输入语句类似，但正文文件读出语句中除通道外，不允许也没有必要用 PROMPT 提示串，因为数据是从正文文件中读出。从正文文件读出时，只能从文件开始读。正文文件读出语句格式为：

INPUT #<通道>:<变量> [{,<变量>}]

LINE INPUT #<通道>:<字串变量$>

第一个语句有两种形式分别对正文文件不同的数据行读出方式：

INPUT #<通道>:<变量> 用于正文文件一行只有一个数据项的情况。由于正文文件写入时，数值自动转变化为数字串，读出时，又自动转换为数值，正文文件允许不同类型数据行混存。因此读出时要求数据与变量类型一致。

INPUT #<通道>:<变量表> 用于正文文件一行有多个数据项，且数据项间有逗号分隔的情况，此时要求数据项数和变量数应相同，类型一致，否则出错。

例 7.6　假设 B 盘上有称 TEXT 的正文文件，其内容为：

```
zhao liuyi,70,67,81,72.6667
weng wu,89.5,75,95,86.5
wang ming,90,85,95,90
hu mingli,89,95,92,92
wang ying,98,85,79,87.3333
li ping,95,85,77,85.6667
zhang lili,70,67,81,72.6667
```

编程用 INPUT 语句读取数据，并显示输出。源程序清单如下：

```
! Read a text file called "text"
LET n$="b:text"
LET ch=1
OPEN #ch:NAME n$,ORGANIZATION text
SET #ch:POINTER begin
SET ZONEWIDTH 12
PRINT "Name","Politics","English",
PRINT "Computer","Average"
LET s,i=0
DO WHILE MORE #ch
    INPUT #ch:name$,p,e,c,a
    PRINT name$,p,e,c,a
    LET s=s+a
    LET i=i+1
LOOP
PRINT
PRINT "Number :";i,"total average =";s/i
CLOSE #ch
END
```

源程序运行结果如下：

```
Name          Politics      English       Computer      Average
zhao liuyi    70            67            81            72.6667
weng wu       89.5          75            95            86.5
wang ming     90            85            95            90
hu mingli     89            95            92            92
wang ying     98            85            79            87.3333
li ping       95            85            77            85.6667
zhang lili    70            67            81            72.6667
Number : 7 total average = 83.8333
```

说明：

① TEXT 正文文件每行有 5 个数据项用逗号分隔，第一个数据项为字符串，其它 4 项是数值。因此，INPUT 语句相应有 N$，P，E，C，A 5 个变量。变量数过多，过少或类型不匹配均会出错。如果文件中同行的数据项间用空格或分号分隔时，则认为一行只有一个字符串数据，亦因变量多、数据项少而出错。无论单变量或用变量表的文件 INPUT 读语句皆不允许数据文件中有空行，否则因数据项少而出错，出错是致命性的，LINE INPUT 文件读出语句可以克服以上缺点。

② 在文件读出操作的循环中文件指针测试函数 MORE 控制循环。当指针未指向文件末尾时，为真值，表明还有数据行可读，继续读取；当指针指向文件末尾时，为假，再无数据可读，结束读操作循环。如果在无数据时再要读出，也将出错。

LINE INPUT 文件读出语句一次读入一行字符串数据，赋予字串变量，对一行是单个数据或空行都能读取。对多数据项组合行，在行读取并赋予字串变量后，然后对字串变量进行分解，如数值型数据，还要作数字字符串到数值的变换。

例 7.7 利用 LINE INPUT 读出 TXTFILE 正文文件数据，并显示输出。

```
! Read a text file called "b:txtfile"
LET n$="b:txtfile"
LET ch=1
OPEN #ch:NAME n$
SET #ch:POINTER begin
LET i=0
DO WHILE MORE #ch
    LINE INPUT #ch:x$
    PRINT x$
    LET i=i+1
LOOP
PRINT "total lines =";i
```

```
CLOSE #ch
END
```

说明：本例完成正文文件一行读入，原样显示输出。对字符行中多个数据未作任何处理，多数据项分离请看例 7.9。

(3) 使用打印机

True BASIC 将外部设备视为系统的正文文件，例如计算机的控制台（键盘和屏幕）看作可读写的正文文件，并指定使用 #0 通道（可缺省）。它始终打开，不能关闭，因此程序运行结果总是在屏幕输出。打印机视为只写的正文文件，为了将运行结果在打印机输出，首先应该打开打印机。打印机打开语句格式为：

OPEN #<通道>:PRINTER

其中 PRINTER 是系统保留字，通道是数值表达式的值，自动四舍五入取整，范围是 1～999 的正整数。该语句不能有任何可选项及参数，用来打开打印机。

在打印打开后，正文文件写语句可用于打印机打印输出：

PRINT #<通道>:[<表达式>][{,/;<表达式>}]

如果将屏幕看作 #0 的正文文件，则它可作为通用的输出语句，取决于通道，如果是与打印机相联的通道，则在打印机打印输出；希望在屏幕输出时，通道是 #0；是正文文件通道，则将表达式的值写入正文文件存储。采用人机对话方法将输出设备的通道赋于变量，则同一程序无须修改就可灵活控制在屏幕显示或在打印机打印输出，见例 7.8。

例 7.8 通用格式 PRINT 语句应用实例

```
! Sample for output selection
PRINT "display(0) or print (1-999):";!
INTUT oc        !
IF oc<>0 THEN OPEN #oc:PRINTER   !
LET n$="b:txtfile"
LET ch=1
OPEN #ch:NAME n$
SET #ch:POINTER begin
LET i=0
DO WHILE MORE #ch
   LINE INPUT #ch;x$
   PRINT #oc;x$              !
   LET i=i+1
LOOP
PRINT "total lines =";i
CLOSE #ch
IF oc<>0 THEN CLOSE #oc      !
END
```

说明：本程序由在例 7.7 中插入 3 个语句及所有 PRINT 语句改造为通用格式构成。它完成打印或显示对话及判断用户是否选择打印输出，如果是，打开打印机，PRINT 语句在打印机输出，打印完成后关闭打印机；如果不是，则 PRINT 语句在屏幕显示输出。同一程序可由用户控制程序运行结果的输出设备。

(4) 正文文件应用实例

例 7.9 利用 B 盘上已存在的学生成绩正文文件 TXTFILE，编制学习成绩管理程序，要求程序具有数据添加、数据读出、求总平成绩和退出运行、返回系统等项功能。

```
! Read / Write a text file called "txtfile"
SUB read
  SET #ch:POINTER BEGIN
  LET i,s=0
  DO WHILE MORE #ch
    LINE INPUT #ch;x$
    PRINT x$
    LET ys=x$[(5-1)*m+1:5*m]
    LET t=VAL(TRIM$(YS))
    let s=s+t
    LET i=i+1
  LOOP
  PRINT "total=";s
  PRINT "i=";i,"s=";s/i
END SUB
SUB write
    SET #ch:POINTER END
    SET ZONEWIDTH m+1
    LET i=1
    DO
      INPUT PROMPT "Name:" ; name$
      IF name$="stop" THEN EXIT DO
      PRINT "more data p,e,c:";
      INPUT p,e,c
    PRINT #ch:name$,p,e,c,
    LET a=(p+e+c)/3
    PRINT #ch: a
  LOOP
END SUB
! main program
LET m=12
LET n$="b:txtfile"
LET ch=1
OPEN #ch:NAME n$
ASK #ch:ZONEWIDTH m
DO
  PRINT "read(r) or write(w) or quit(q)?";
  INPUT a$
  IF a$="r" THEN
    CALL read
  ELSE
    IF a$="w" THEN
      CALL write
    ELSE
      IF a$="q" THEN EXIT DO
    END IF
  END IF
LOOP
CLOSE #ch
END
```

说明：

① 该程序有两个内部子程序 WRITE 和 READ，分别完成正文文件 TXFFILE 的数据添加和读出求总平均值处理。由于主程序和子程序是一个程序单位，在主程序中打开的 #CH 是全程通道，使正文文件 TXTFILE 为程序内的模块共享，正文文件的读、写操作完全由文件指针控制。

② 正文文件 TXTFILE一行有 5 个域的数据，其中第 5 域是每个学生的平均成绩。LINE INPUT 将一行字串读入赋予 X$，利用主程序中查询的正文文件域宽全程变量 m，截取 X$ 第 5 域，数值还原后获得每个学生的平均成绩。

③ 如果希望文件读出后每个学生的成绩按域列表显示，则需将 X$ 分解处理，显示输出的域宽可以设置与文件域宽相同或不同的合适值。例如设置成 m+1，保证文件中宽度为 m 的数据按域格式显示。

例 7.10 在 B 盘上有一个称 DICT 的字典正文文件，它存储着用户收集的英文单词和词组。运行以下程序时，用户可在键盘上键入一个单词或词组，程序将告知它是否已被收集在其中，如果没有，它将按用户意图将其添加到字典的末尾。

```
! Ask for words from user,and look them up in
! a dicionary If not found, add them to
! the end of the file.
!
LET n$="b:dict.txt"
OPEN #1:NAME n$,ORGANIZATION text
DO
  PRINT "ask a word";
  PRINT "(press return key for end):";
  LINE INPUT word$
  IF word$<>"" THEN
    CALL lookup(#1,word$,lp)
  IF lp=0 then
      PRINT "not found!"
      CALL add(#1,word$)
    ELSE
```

```
      PRINT "found"
    END IF
  END IF
LOOP UNTIL word $ =“”
END
SUB LOOKUP(#99,w$ ,lp)
  SET #99:POINTER BEGIN
  LET w$ =LCASE$ (w$ )
  LET lp=0
  DO WHILE MORE #99
    INPUT #99:d$
    IF w$ =d$ THEN
      LET lp=1
      EXIT DO
    END IF
  LOOP
END SUB
SUB ADD(#99,w$ )
  PRINT "add it to the dictionary (y/n)?";
  INPUT an$
  LET an$ =LCASE$ (an$[1:1])
  IF an$ <>"y" THEN EXIT SUB
     SET #99:POINTER END
     PRINT #99:LCASE$ (w$ )
END SUB
```

说明：

① 本例假设 B 盘 DICT 的正文文件已经建立，如果 DICT 不存在，程序出错停机。为此，在程序运行前首先在建立 DICT 文件，或编写 DICT 文件建立程序，或直接用系统命令建立。可以。建立包含若干单词或词组的 DICT 文件，也可以建立一个有名无实的空文件，随着程序运行，它将不断扩展其内容。

②本例中有 3 个程序单位：主程序、LOOKUP 和 ADD 两个外部子程序。LOOKUP 子程序实现键盘输入的英语单词或词组在字典文件中查寻任务。ADD 子程序实现单词或词组的添加任务。字典文件 DICT 利用通道参数实现程序单位间的共享。

③ 外部子程序 LOOKUP 有三个形式参数：通道参数 #99 和 W$，主程序经它们传递字典文件和要查找的单词；LP 是变量参数，给主程序返回查寻的标志。子程序从字典文件首部开始按顺序逐行读出，一行是一个单词或词组与键入单词比较，如果查找成功，LP 置 1；查寻不成功，即字典文件中原没有该单词或词组时，标志 LP 置 0。

④ 外部子程序 ADD 有两个参数：#99，W$，它将新单词或词组添加在文件的末尾。

⑤ 参数都有局部变量的特性，虽然有两个子程序都有通道参数 #99 和 W$，但它们在各自的程序单位有效，各不相同，只取决于调用的实参。

7.5 记录文件

正文文件数据的读、写操作，只能由文件的开始按行顺序进行，就是查询其中一个数据，也可能要从开始到末尾全部读出文件内容，才能确定数据是否存在，访问速度慢；如果不用系统命令，在程序控制下插入、删除或修改数据，都只能删除原文件内容后重写，很不方便。

记录文件是记录的集合，记录格式随机访问式的文件，记录是文件读、写访问的单位。文件中每个记录具有相同的长度，长度用字节数表示，称定长记录。一个记录可以只包含一个字段(数据项)，称单字段记录；也可以包含多个字段，称多字段记录。字段数据可以是字符串，或是数值。对单字段记录文件，可以是两种数据类型记录的混合；对多字段记录，允许不同类型字段经变换混存于同一记录中。True BASIC 用 1 开始的正整数对所有的记录编号，文件指针指向的记录称当前记录，它指出当前文件操作的记录。文件指针设置语句可将文件

任何一个记录指定为当前记录，从而可以随机访问文件中的任何一个记录。记录文件的数据采用内部存储形式，文件内容不能在屏幕上显示，也不可能用系统命令编辑、修改。记录文件的随机访问特性，同时数据不需转换成可显示形式，访问速度比正文文件快得多。

记录文件用 READ 和 WRITE 语句实现读、写操作，文件长度可达 10 亿字节，可适应各种计算机的应用。

(1) 记录长度和文件指针设置

记录文件由定长的记录集合组成，记录长度以字节为单位，允许最大长度是 2^{24}(约16M)字节。字符串型字段数据的每个字符占用 1 个字节，数值型字段占 8 个字节。在一个单字段记录文件中，允许字串型记录和数值型记录组合存储。记录文件建立时应指定记录长度，它可以在打开文件语句中由 RECSIZE 可选项指定，或用记录长度设置语句设定。设置语句格式为：

SET #<通道>：RECSIZE <数值表达式>

其中表达式的值是指定记录长度的字节数，自动四舍五入取整。该语句在记录文件建立时，设置记录长度，亦可用作空文件的记录长度重新定义。在访问已存在的非空的记录文件时，不允许用它设置记录长度，否则出错，终止程序运行。

记录文件打开后，指针指向 1 号记录，在当前记录每进行一次读、写操作，指针自动指向下一个记录。在最后一个记录被访问后，指针指向文件末尾(END)，文件测试函数 END #< 通道>为真值。

文件指针设置语句格式为：

SET #<通道>：POINTER <参数>

或　RESET #<通道>：<参数>

其中参数指出文件指针设定的位置，参数可用 BEGIN 指针指向 1 号记录；END 指向文件末尾；SAME 是指针在操作后停留，不下移；NEXT 指向当前记录的下一个记录；或直接用数值表达式指出记录号。参数的意义见图 7.1，图中箭头表示当前记录。

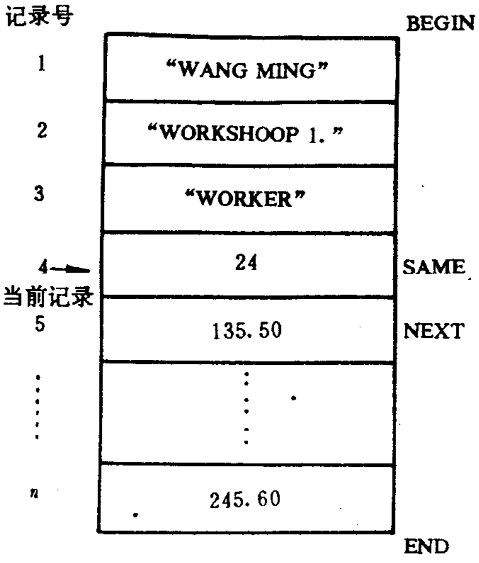


图 7.1 文件指针及参数

(2) 记录文件写入操作

在记录文件以只写(OUTPUT) 方式或读、写 (OUTIN) 方式打开时，WRITE 语句将数据写入记录文件的当前记录。写入语句格式为：

WRITE #<通道>:<表达式> [{,<表达式>}]

其中表达式是字符串型，也可以是数值型。一个 WRITE 语句将一个表达式的数据写入当前记录，也可以将多个表达式的数据，每个数据作为一个记录，从当前记录开始依次写入文件。本节和下节只介绍单字段记录的写入、读出操作。由于一个记录是一个字段，允许不同类型记录混存于文件中。在数据写入时，为防止数据出错，应以最长的数据长度设为记录长度；同时，在记录读出时，必须注意变量与数据类型匹配，否则出错。WRITE 语句经文件指针设置可以重写某个记录，或在文件末尾添加记录。

例 7.11　在 B 盘上建立称 recfile 的记录文件存储职工的姓名、工作部门、职称、年龄和

工资情况。

```
! Create a record file called "recfile"
LET n $ ="a;recfile1"
LET ch=1
OPEN #ch;NAME n $ ,ORGANIZATION record,CREATE newold,RECSIZE 12
RESET #ch;END
DO
    INPUT PROMPT "Name ;";name $
    IF LCASE $ (name $ )="stop" THEN EXIT DO
    INPUT PROMPT "more data ;";d $ , t $ ,a,s
    WRITE #ch;name $ ,d $ ,t $ ,a,s
LOOP
CLOSE #ch
END
```

① 程序以 NEWOLD 方式打开，并将文件指针指向末尾，既适用于新建 recfile 记录文件，又能再次使用作记录添加。

② 例中每个职工的情况分别用 5 个记录存储，姓名、部门和职称是字符型字段，年龄和工资是数值型字段，并按最长的字段数据设置记录长度是 12 个字节。

(3) 记录文件读出操作

记录文件数据不可显示，文件中数据必须由程序控制读出再显示。当记录文件以只读(INPUT) 或读、写(OUTIN)方式打开时，语句 READ 读取文件当前记录的数据。读出语句格式为：

READ #<通道>:变量 [{,<变量>}]

该语句允许只读取当前记录的数据，赋予 READ 语句中的单个变量；也允许从当前记录开始读取相同记录的数据，依次赋予多个变量。在单字段记录读出时，要求变量与对应读出数据类型一致，否则出错；如果指针已经指向文件末尾，还要继续读出，也会出错。

MAT READ #<通道>:数组[{,<数组>}]

该语句从当前记录开始读出数据，依次作为数组元素。允许读入多个数组。

例 7.12 读出 B 盘上例建立的 RECFILE 记录文件，随机读出某个记录数据，并在屏幕上显示输出。

```
! Read and display a record file called "fecfile"
LET n $ ="a;recfile"
LET ch=1
OPEN #ch;NAME n $ ,ORGANIZATION record
SET ZONEWIDTH 13
PRINT
PRINT "Name" ,"department" ,"title" ,"age" ,
PRINT "salary"
PRINT
DO WHILE MORE #ch
    READ #ch;name $ ,d $ ,t $ ,a,s
    PRINT name $ ,ds,ts,a,s
LOOP
ASK #ch;FILESIZE fs
DO
    PRINT "enter recno. ";"(1—";fs;")";
    INPUT PROMPT " for check;";no
    IF no<=0 OR no>fs THEN EXIT DO
    RESET #ch;RECORD no
    WHEN EXCEPTION IN
        READ #ch;m $
        PRINT m $
```

```
USE
    PRINT NUM(m$)
END WHEN
LOOP
CLOSE #ch
END
```

说明：

① 记录文件 RCFIFE 已经存在，所以 OPEN 语句中文件类型和记录长度这两个可选项实际上可以省略，按原文件类型及记录长度访问该文件。如果可选项不省略，则必须指定与原文件相符的类型和记录长度，否则出错。

② RECFILE 是单字段记录文件，并且字符串型和数据型记录混存。因此在随机读出某个记录的数据时，读出类型如何确定？为了保证程序正确运行，在随机读出的 DO/LOOP 循环中嵌套了 WHEN 错误处理结构，在保护模块中，记录数据作字符串型读出赋予变量 m$，显示输出；如果不是字符串型字段，则在处理模块中由 NUM 函数作数值变换，再显示输出。无论记录是字符串或是数值，程序都能正确运行。

错误处理结构可以屏蔽各种故障的发生，确保程序正常运行。如果将记录文件读出语句提出错误处理结构之外，在读取数值型记录时，将出现“Data item isn’ t a string”的错误。由于记录文件的数据以不可见形式存储，在处理模块中，必须用 NUM 函数变换成数值才能显示输出，如果用 VAL 函数，则将出现 “VAL string isn’ t a proper number”的错误。不允许不作变换直接显示输出，否则也将出错。

③ 列表显示输出的域宽设置，一般应大于或等于记录文件的记录长度，以保证数据不会超出域宽，破坏列表。

④ 程序利用文件大小查询语句，由 fs 返回文件记录总数，在要求输入查寻记录号时，提示可查记录范围，访问超出范围的记录，程序出错，终止程序运行。

(4) 多字段记录

在实际应用中，数据文件总有一系列不同类型的数据来描述实体的特性，要求用多字段记录。单字段记录文件，存在两个问题：一是由于记录数据类型不一，数据长度也可能相差甚多，为保证记录文件随机访问的特性，要求所有记录统一成定长的记录存储，造成存储空间的浪费；二是在记录读出时，要求变量与数据类型匹配，正如上例所见，不小心处理就会出错，同时，记录寻找也不方便，例如第一个职工的姓名，工资分别在第 1,5 记录，而第二个职工则在第 6,10 记录，难以管理。

多字段记录是将实体描述的所有字段合并，存储在一个字符串型的记录中。每个字段都有规定的长度，记录长度是各字段长度的总和。由于它是定长字符串型记录，保证其随机访问的特性。在记录读出时，程序按各字段长度从字符串变量中分离和处理。因此，在多字段记录的读、写操作中，重要的是解决类型变换、字段合并和分解的问题。

① 数值型字段变换及还原。True BASIC 提供用于记录文件中数值到数字字符串的变换和还原的两个函数 NUM$ 和 NUM。将数值变换成字符串函数调用格式为：

NUM$ (<数值表达式>)

它返回表达式值的 IEEE 标准，8 个字节长度的数字字符串。IEEE 标准的数字字符串在所有的机器上通用，变换的数值范围是 1E－307 到 1E307，精度可达 15 位。变换后的数字字符串不能在屏幕上显示，主要用于写入记录文件。

NUM$ 函数变换换后的数字字符串用 NUM 函数还原，该函数的调用格式为：

NUM（<字符串表达式>）

它将 NUM$ 函数建立的数字字符串还原成数值，主要用于记录文件读出。

② 字段合并。为将类型和长度不一的各个相关字段合并成一个定长记录，首先数值型字段，在合并前利用 NUM$ 函数将它变换成标准的 IEEE 8个字节的字符串。其次是同一字段统一长度，按该字段最长的数据作标准，规定长度。为了保证字符串在定长的字段中按左侧调整设置，使用一个自定义函数：

```
DEF fixlen$(a$,n)=(a$ & REPEAT$(" ",n))[1:n]
```

自定义函数 fixlen$ 有两个值参 a$ 和 n，其中 a$ 表示字符串型字段，n 是该字段以字节为单位的统一长度。自定义函数返回长度为 n 的字符串，在 a$ 长度不足 n 个字节时，其后用空格符补足。在所有字段均已转变成定长的字符串后，连接构成一个字符串型定长记录，记录长度是各字段长度的总和。

③ 字段数据分离。多字段记录文件可随机读出任何一个记录，赋予字符串变量。记录读出后，利用字符串截取操作，按各字段规定的长度从读取的数据中分离各个字段，对原数值型字段再用 NUM 函数还原。此后，程序可显示，或按要求处理记录的各字段数据。

(5)记录文件应用实例

例 7.13　用多字段记录文件存储某单位职工情况，演示字段变换合并和分解及显示输出。

```
! Create and read a mult-fields record file called "mrecfile"
DEF fixlen$(a$,n)=(a$ & REPEAT$(" ",n))[1:n]
LET n$="a:mrecfile"
LET ch=1
OPEN #ch:NAME n$,ORGANIZATION RECORD,CREATE NEWOLD,RECSIZE 50
RESET #ch:END
DO
    INPUT PROMPT "Name :";name$
    IF LCASE$(name$)="stop" THEN EXIT DO
    INPUT PROMPT "more data d$,t$,a,s :";d$,t$,a,s
    LET p$=fixlen$(name$,10) & fixlen$(d$,12) & fixlen$(t$,12)
    LET p$=p$ & NUM$(a) & NUM$(s)
    WRITE #ch:p$
LOOP
RESET #ch:BEGIN
SET ZONEWIDTH 13
PRINT
PRINT "Name","department","title","age","salary"
PRINT
DO WHILE MORE #ch
   READ #ch:st$
   CALL unpack(st$,n$,10,d$,12,t$,12,a,s)
   PRINT n$,d$,t$,a,s
LOOP
```

```
CLOSE #ch
END
SUB UNPACK(st$,n$,n1,d$,nd,t$,nt,a,s)
    LET n$=st$[1:n1]
    LET n2=n1+nd
    LET d$=st$[n1+1:n2]
    LET n3=n2+nt
    LET t$=st$[n2+1:n3]
    LET n4=n3+8
    LET a=NUM(st$[n3+1:n4])
    LET n5=n4+8
    LET s=NUM(st$[n4+1:n5])
END SUB
```

7.6 字节文件

字节文件是以字节存取的顺序文件。它的单字符读取能力适合用于自由格式数据，使之能处理由其它程序（如字处理、排版处理程序及其它应用程序）建立的数据文件。由于字节文件适应性强，在设计一个读取自由格式数据程序时，画出一个允许输入数据结构的语法图十分有用，它不仅能简化程序设计，而且有助于其它用户为它准备数据。True BASIC 语言的字节文件不仅可以读写单个字符，且允许读取指定长度的字节流（字符串）。此外，它有顺序文件的特性，由文件指针控制读或添加操作。字节文件打开后亦由 WRITE 语句写入，READ 语句读出。

(1) 字节文件写操作

在字节文件以只写或读写方式打开时，允许向新建文件或已删除内容的文件写入字节流，对已存在的文件允许在文件末尾添加字节数据。只有字符串表达式代表的字符才能写入字节文件，并按文件指针所指顺序存储，每写入一个字符，指针后移一个字节。字节文件写入语句格式为：

WRITE #<通道>:<字符串表达式> [{,<字符串表达式>}]

其中字符串表达式可以是单个字符，或字串变量，或表达式代表的字符串。字节文件没有修改操作，允许添加或重写。

(2) 字节文件读操作

字节文件读出数据只能赋予字符串变量，可以逐个字符读出，或按指定的字节长度数读出。因此，字节文件读取语句格式为：

READ #<通道>,BYTE <数值表达式>:<变量$> [{,<变量$>}]

它的等效格式为：

SET #<通道>: RECSIZE <数值表达式>

READ #<通道>:<变量$> [{,<变量$>}]

它从文件指针所指字节开始读取字节流，直到读取<数值表达式>值指定的字节数，并赋给字符串变量，指针后移相应字节数。如果尚未读出足够的字符，文件指针已达文件末尾时，将

出错，中止读出操作。

在现实世界中的正文不仅仅是字符流，而是用各种方式构造的正文文件，例如程序、数据和散文，通常是按行和页构造。程序用 PRINT 语句或 LINE INPUT 语句产生行结构的正文文件。由于正文文件的行结构以回车符结尾，因此，程序利用字节文件可以十分简单地读取和识别这一结构。同样，记录是固定长度结构的字符流数据，记录文件也可作字节流读取，由程序按给定的结构要求识别和处理。任何基本类型构造的数据都可以看作有结构要求的字符流，可用字节文件存储，从字节文件中读出，按已知的结构复原，再由程序加工处理。所以字节文件可适用于任何数据结构文件，它具有更强的适应性和实用性。本节例 7.15 演示程序为用字节文件处理正文文件的实例。

例 7.14　统计一篇英文短文中冠词 "THE" 和 "a","an" 出现的频数和百分率。该英文短文存储在“P9－UT”的数据文件中。

```
! Read a program called "p9—ut" (e7—11)
SUB testm
    SELECT CASE m$
    CASE "a","an"
         LET na=na+1
    CASE "the"
      LET nt=nt+1
    CASE ELSE
         LET nu=nu+1
    END SELECT
END SUB
! main program
LET n$="a:p9—ut"
LET nu,na,nt=0
LET ch=1
OPEN #ch:NAME n$,ORGANIZATION byte
LET m$=“”
DO WHILE MORE #ch
  READ #ch,BYTES 1:s$
  LET s$=LCASE$(s$)
  IF s$=CHR$(13) OR s$=" " OR s$="."
OR s$="," THEN
    PRINT m$;" ";
    CALL testm
    LET m$=""
  ELSE
    LET m$=m$ & s$
  END IF
LOOP
PRINT
PRINT "total words:";nu,"a,an:";na;
PRINT na* 100/nu,"the:";nt;nt*100/nu
CLOSE #ch
END
```

说明：短文是行结构的正文，假定行中的英文单词用空格、逗号、句号或回车等符号分隔，程序按此规则捡出英文单词，统计冠词出现频度和占全文单词的百分数。

小　结

文件是存储在磁盘上的信息集合，磁盘存储容量大，并能永久性保存信息，所以文件一旦建立可多次使用。

按存储信息类型可分为程序文件和数据文件两类。程序文件存储程序，它又分为源程序文件和编译文件两种。程序文件由系统管理，受系统命令控制。数据文件保存程序处理用的数据资源或处理结果，由程序控制管理。

程序操作数据文件的过程：打开文件，文件处理和关闭文件。

OPEN 语句指定打开文件及其通道，指定访问方式或核对文件的建立方式及文件类型，

文件打开后指针指向文件开始，程序按指定方式访问打开的文件，处理数据。访问未经打开文件或核对文件参数不符，都将出现致命性错误。

Close 语句关闭文件。True BASIC 有三种数据文件：正文文件是字符行结构的顺序文件，字符是可显示的 ASCII 码，因此，利用系统命令建立、编辑、监视正文文件十分简便；程序用 INPUT，LINE INPUT 和 PRINT 语句控制正文文件读、写，正文文件只能从开始读出，在末尾添加。因此，可由文件指针设置，控制正文文件读、写操作。

记录文件是记录结构的随机文件，它由固定长度记录集合组成。按记录包含的数据项，可分为单字段记录文件和多字段记录文件。

在单字段记录中只含一个数据项，允许有不同数据类型记录混存，但记录长度仍是固定统一，因此存储空间浪费较大，程序管理不方便。

多字段记录中，一个记录可包含多个数据项。多个数据项经变换、合并，构成统一长度的字串型记录，读出时再分离、还原。与单字段记录相比，程序管理简单方便，节省存储空间。

记录文件具有随机访问特性，文件数据不可显示，访问速度要比正文文件快。语句 WRITE ，READ 控制记录读、写操作。

字节文件是字符流的顺序文件，类似于正文文件，但无行结构限制，可方便地读入行结构或记录结构型数据，由程序识别和处理各种结构的数据，所以字节文件更为灵活，适用于任意数据结构的读写操作。

程序用 WRITE ，READ 语句控制字节文件读写。

习　题

1. 什么是程序文件？什么是数据文件？

2. True BASIC 数据文件有几种，它们各有什么特点？各自适应什么应用场合？

3. 将一个源程序作为正文文件，编写程序打印该源程序。

4. 以 10 个职工的情况（包括编号，姓名，年龄，性别，民族，文化程度，政治面貌等）作为数据文件，编写一个简单管理程序，要求有建立、添加、查寻显示、输出等项功能。

? 用正文文件建立实现。

! 用单字段记录文件实现，并增加记录随机修改功能。

: 用多字段记录文件实现，并增加记录字段随机访问功能。

' 用字节文件实现。

5. 将上题建立的职工情况正文文件读出，编程由用户选择决定输出是显示、打印或存储于另一个正文文件中。

6. 有两个文件 a，b 分别存储两个班的学生姓名、入学成绩。试编程序，分别按姓名排序，合并两个文件，并要求合并的文件按姓名排序。

7. 设计程序在按姓名升序排列的入学成绩数据文件中，查寻某个同学的入学成绩。

8. 设计程序将某个 True BASIC 的源程序按字节文件读入、处理，要求按原样输出该源程序清单。

第8章　图形与声音

据统计,人类90%以上的信息依靠视觉和听觉器官获得,可见图和声是多么重要的信息媒体。目前,由于图、声信息要求存储容量太大,识别条件不充分,计算机还不可能自动识别,但以扩大计算机用户层为目的,作为图、声识别过渡的多媒体技术受到人们的青睐。与文字、数字信息相比,图、声信息最突出的优点是:信息量大,形象,易于人们理解和记忆。特别是图,正如俗话所说,“一幅画抵得上千言万语”。因此计算机作图和发声技术深受用户欢迎。

图(PICTURE)包括图形(GRAPHICS)和图像(IMAGES)两类。图像是客观世界的反映,由遥感、摄像等手段获取,由计算机处理,称计算机图像学;图形信息来自主观世界,经人们构思,由计算机产生,称计算机图形学。目前,计算机图形学已经广泛地应用在各个领域,如计算机辅助设计(CAD)和辅助制造(CAM)、仿真、地图绘制和地理学信息处理、管理系统中的数据分析和决策、辅助教学(CAI)、办公室自动化(OA)、游戏及动画制作等等。

本章介绍True BASIC的作图和发声语句,以及最基本的作图和发声技术。

8.1　图形环境与作图语句

True BASIC提供了与显示器像素无关的作图语句,不依赖某种图形卡画点、线、圆及标注图形文字的能力。

8.1.1　显示方式和模式设置

(1)文本方式、图形方式

计算机的显示器相当一张稿纸,在程序控制下,可以在上面写字,也可以作图。显示器采用不同的方式显示文本和图形,前者称文本方式,后者称图形方式。在文本方式下,字符是显示的基本单位,例如,一屏可显示25行,每行80个西文字符。文本显示方式用于书写,显示行结构的源程序、数据和散文,或由程序PRINT语句输出数据和文本。在文本方式下不能作图;在图形方式下,像素(Pixel)是显示的基本单位,一个像素是显示器上的一个点,随显示器的分辨率不同,显示器可能是640×200、320×200、640×350或1024×768等个像素组成的点阵。由于像素排列紧密,某些点的组合显示构成图形。在图形方式下,由程序PLOT语句作图及标注图形文字,所以,图形方式实际是图、文兼容的显示方式。

按IBM-PC系统配置的显示接口卡和显示器不同,具有8种不同特性的显示模式,见表8.1。对单色显示卡和显示器只能使用(“MONO”)模式,只能显示单色的文本。如果是彩色图形接口卡(CGA)和彩色显示器,可以使用其余7种模式,例如,“80”模式,一屏显示25行,每行80个西文字符多彩的文本;“GRAPHICS”模式,可用于分辨率为340×200图形方式,显示图、文并茂,彩色丰富的图形,也可以用于一屏10行,每行40个西文字符的彩色文本显示方式;“HIRES”模式,可用于分辨率为640×200单彩色图形方式,也可以用于一屏25行,每行80个西文字符的彩色文本显示方式。2.03版本True BASIC还支持单色图形接

口卡(HERCULES)、更高档的彩色图形卡(EGA、VGA、TVGA)和显示器,支持“EGAHIRES”模式,可用于分辨率为 640×350 彩色图形方式,也可以用于一屏 25 行,每行 80 个中、西文字符的彩色文本显示方式。

表 8.1　IBM—PC 计算机的显示模式

模式名	文本行宽(列)	字符集%	字符闪烁	彩色	图形分下率	光标闪烁
MONO	80	256	有	无	/	有
40	40	256	有	有	/	有
BW40	40	256	有	有	/	有
80	80	256	有	有	/	有
BW80	80	256	有	有	/	有
GRAPHICS#	40	128	无	有	320×200	无
BWGRAPHICS	40	128	无	有	320×200	无
HIRHS	80	128	无	有	640×200	无
* HERCULES	80	256	无	无	720×348	无
* EGAMONO	80	256	有	无	640×350	有
* EGAHIRES	80	256	无	有	640×350	无

%:参看附录 1 中有关 IBM—PC 字符集

#:2.03 版本中称 MEDRES 模式;*:2.03 版本支持的模式

(2)显示模式设置

对配置彩色图形卡和显示器的系统,当 True BASIC 启动后,自动设置成“80”模式。根据实际需要,可以用模式设置系统命令,或在程序中用模式设置语句,设置显示模式。

背景窗命令行模式设置命令格式为:

OK. MODE < 模式名 >

其中,模式名不能包括在双引号中。

程序中模式设置语句格式为:

SET MODE < 字符串表达式 >

其中,字串表达式可以是定义为特定模式的字串变量,也可以是包括在双引号中的模式名,模式名字串的字符大小写等效。例如,

SET MODE "HIRES"	设置成 HIRES 模式
SET MODE "80"	设置成 80 模式
SET MODE "GRAPHICS"	设置成 GRAPHICS 模式

如果设置了新模式,IBM—PC 机就自动清屏,因此,无论如何也不能使屏幕的一部分是一种显示模式,而另一部分为另一种模式。设置的显示模式一直保持,直到下次模式设置,或程序运行完毕,自动恢复 True BASIC 的初始模式。如果在模式设置时,使用了 True BASIC 不能辨识的模式,则系统以模式“40”替代。

在任何时候,程序都可以利用模式查询语句了解当前使用的显示模式。模式查询语句格式为:

ASK MODE < 字符串变量 >

语句将当前模式名以大写字符形式,送存字符串变量,返回程序。

8.1.2 窗口和图形窗口坐标设置

许多图的范围从理论上讲是无穷的，事实上，用户只对某个特定范围的部分感兴趣，所以往往取一个矩形区加以限制，它相当于照相机的取景器。计算机在显示器的屏幕上作图，屏幕上的矩形输出区称窗口（WINDOW）。

True BAISC 把窗口看成是只写的特殊文件。可以将整个显示器的屏幕作为一个窗口，也可以将显示器的屏幕划分为多个窗口。文本方式的显示窗口，称文本窗口；图形方式的显示窗口，称图形窗口。在屏幕上有多个窗口时，允许两种窗口同时并存，但整个屏幕只可能为一种显示模式。例如，True BASIC 启动时，整个屏幕为“80”模式，划分为两个文本窗口，分别称编辑窗口和背景窗口。

图形窗口是一个二维的直角坐标系，坐标原点位于屏幕窗口的左下角，水平坐标轴 X 的正方向是自左向右，纵坐标轴 Y 的正方向是自下而上，窗口中任意一点可用 X，Y 两个坐标定位。在配置 CGA 卡和彩显的系统中，开始运行作图程序时，True BASIC 自动置成“GRAPHICS”模式，并将显示器的整个屏幕作为一个图形窗口打开，称缺省图形窗口。缺省图形窗口是通道号为 #0，总是打开的单窗口，系统自动设置它的坐标，水平坐标最小值 $X_{min}=0$，最大值 $X_{max}=1$，纵坐标的最小值 $Y_{min}=0$，最大值 $Y_{max}=1$。

图形窗口的坐标可以无量纲，也可以采用实际单位，如年份、公里、秒、牛顿、人民币元等作为量纲。程序可以通过语句改变窗口的坐标，窗口坐标设置语句格式为：

```
SET WINDOW <Xmin>,<Xmax>,<Ymin>,<Ymax>
```

语句设置窗口坐标，坐标使用实际单位，其中 X_{min}，X_{max}，Y_{min}，Y_{max} 为数值表达式，窗口的横坐标最小值和最大值由数字表达式 X_{min}，X_{max} 决定，纵坐标的最小值和最大值由 Y_{min} 和 Y_{max} 决定。根据窗口坐标设置，坐标原点可能在屏幕左下角，也可能在窗口中，也有可能在窗口外。缺省图形窗口亦可以通过坐标设置语句改变窗口坐标。

利用查询语句可以了解窗口坐标设置，查询语句格式为：

```
ASK WINDOW <X1>,<X2>,<Y1>,<Y2>
```

其中 X1、X2、Y1、Y2 为数值变量，该语句将当前窗口的最小横、纵坐标值送数值变量 X1、Y1，最大横、纵坐标值送变量 X2、Y2，程序利用这 4 个变量了解当前窗口设置的坐标值。

人们可以根据实际需要设置窗口坐标，使用实际坐标单位值在窗口中作图或标注文字，与显示器无关，也无需考虑屏幕像素和窗口坐标的映射关系。

8.1.3 彩色设置

在配有 CGA 彩色图形卡和彩色显示器的计算机上，具有彩色和图形功能，在图形程序运行时，系统自动设置成 GRAPHCS 模式，为 320×200 分辨率的图形显示方式。在彩色未加设置时，窗口的背景（相当图纸）是黑色，前景（相当图笔）是绿色，此时，相当于在黑色的图纸上用绿笔作图。在 GRAPHCS 图形模式下，作图的前景色和窗口的背景色均允许程序设置、改变。

（1）前景色设置

在 GRAPHCS 图形模式下，IBM－PC 机提供了两块调色板，每块调色板包含三种不同的颜色，加窗口背景色，共七种可供选择的前景色，见表 8.2。任何时刻只能选择使用其中一种作前景色，并且，窗口中的图形只可能是同一调色板上的彩色构成。如果作图过程中，更换、选择另一块调色板中的彩色作前景色时，则窗口中图形颜色将自动改变成当前调色板包含

的对应颜色，二者的对应关系如下：

GREEN(绿) ↔ CYAN(青兰)
RED (红) ↔ MAGENTA(洋红)
YELLOW(黄) ↔ WHITE(白)

表 8.2 GRAPHICS 图形模式前景可供选用的颜色

调色板 1		调色板 2	
编号	色名	编号	色名
1	GREEN	4	CYAN
2	RED	5	MAGENTA
3	BROWN *	6	WHITE
0	BACKGROUND		

* 注：有的计算机可能出现黄色，不是棕色

例如，设置调色板 1 中的红色作前景色，在窗口中画一条直线，又用绿色画了一个三角形，现在要用调色板 2 中的白色画圆，此时，窗口中已有的直线变为青兰色，三角形变为洋红色，刚画的圆是白色，全部变成调色板 2 的颜色。

谁也不会选用背景色作图，因为所作的图看不见。但是，如果选择背景色作画笔，重画窗口中已有的图形，则可以从窗口中“擦除”该图形，赋于人们改修图形的能力。

前景色设置语句格式为：

SET COLOR ＜ 字符串表达式/数值表达式 ＞

其中，字符串表达式实质是利用色名字符串指定的前景色，则色名应包括在双引号中；数值表达式的值是正整数，用编号数指定前景色。前景色设置不影响背景色。

程序可以用查询语句查询当前前景色。查询语句格式：

ASK COLOR ＜ 变量/变量 $ ＞

该语句用字符串变量返回当前前景色的色名，或数值变量返回当前前景色的编号。

注意，两个语句中的“/”表示或，表明每个语句都有两种可能形式。

(2)背景色设置

图形背景色的设置更灵活，有更多的颜色(总共 16 种)可供选择使用，见表 8.3 。其中编号 8～13 和 15 无色名，编号 8～15 的颜色与编号 0～7 对应相同，只是亮度不同。

表 8.3 GRAPHICS 图形模式背景可供选用的颜色

编号	色 名	说明	编号	色名	说明
0	BLACK	黑	8	/	灰
1	BLUE	兰	9	/	亮兰
2	GREEN	录	10	/	亮录
3	CYAN	青兰	11	/	亮青兰
4	RED	红	12	/	亮红
5	MAGENTA	洋红	13	/	亮洋红
6	BROWN *	棕	14	YELLOW *	黄
7	WHITE	白	15	/	亮白

程序在任何时刻都可以改变当前窗口的背景色。背景色设置语句格式为：

SET BACK[GROUND COLOR]〈字串表达式/数值表达式〉

程序查询当前背景色的语句格式为：

ASK BACK[GROUND COLOR] < 变量/变量$ >

背景色设置或查询语句中，关键字可以用全称 SET/ASK BACKGROUND COLOR，亦可以用简称 SET/ASK BACK，两者等效。语句其它部分意思与前景色中说明相同。

如果背景色改变，当背景色与窗口图形的前景色相同时，则窗口中的图形会消失，被“隐藏”。因为，当背景色更改成与图形前景色不同时，它又会重新出现。

8.1.4 作图语句

背景色或前景色的设置后，利用图形语句 PLOT 可以直接在当前窗口中作图。

(1)画点

画点语句格式为：

PLOT [POINTS:] <X>,<Y>

其中 X,Y 为数值表达式，关键字中 [POINTS:]任选，该语句在当前窗口中，用当前的前景色，在数值表达式 X,Y 代表的窗口水平坐标和纵坐标值处画一个点。如果表达式 X,Y 的值不在窗口中，自动修剪被忽略。

例 8.1 STARS 程序在坐标未加的设置的缺省窗口中，随机产生 X,Y 坐标值，先用绿色作前景在黑色的背景上画一个点，恰似夜空中的一颗星星。PAUSE 1 是暂停 1 秒钟，随后又利用随机函数产生 1～6 随机数，用作色码，选择画下一颗星的前景色，画星星。在画了 10 颗星后，在 16 种背景色中随机选择一种改变背景色，继续画星，直到在键盘上按任意键时，KEY INPUT 函数为真值，退出循环，程序终止执行。在该例中可以看到星星的隐藏、重新再现的现象。

```
! Plot random points,like stars on the sky
!
RANDOMIZE
DO
    LET i=1
    DO
        LET x=rnd
        LET y=rnd
        PLOT x,y
        PAUSE 1
        LET i=i+1
        SET COLOR INT(RND * 6)+1
    LOOP UNTIL i>10
    SET BACK INT(RND * 16)
LOOP UNTIL KEY INPUT
END
```

(2)画直线

画直线语句格式为：

PLOT [LINES:] <X1>,<Y1> [{;[<X2>,<Y2>]}]

其中 X1,Y1,X2,Y2 为数值表达式，该语句在当前窗口中，用当前的前景色画直线或折线(连续画直线)，X1,X2,Y1,Y2 的值代表窗口中某点的水平坐标值和纵坐标值。语句关键字[LINES:]部分可以省略。它有下列几种可能形式：

① PLOT X1,Y1　　画点，无直线定向作用。

② PLOT X1,Y1;　　语句在窗口坐标的 X1,Y1 处画点，该点是直线始点，并直线定向后继点，以后继点为直线终点画直线。PLOT 语句中的分号“;”称直线定向符，类似于文本

输出 PRINT 语句的分号作用。

③ PLOT X1,Y1;X2,Y2　它以 X1,Y1 为始点，以 X2,Y2 为终点，画直线，并去消直线定向作用。

④ PLOT X1,Y1;X2,Y2;　它以 X1,Y1 为始点，以 X2,Y2 为终点画直线，并直线定向于后继点。即它以 X2,Y2 为始点，后继点为终点，连续画直线，实际上是画折线。

⑤ PLOT　去消直线定向作用。

例 8.2　TRIANGLES 程序按 GRAPHICS 图形模式的像素点阵(320×200)，设置缺省窗口坐标。首先在黑色背景上随机产生三个点，利用直线定向作用，以绿色画第一个三角形，然后去消直线定向作用，随机更换前景色和窗口背景色，继续画三角形，直到键盘上按任意键，终止程序运行。

```
! Random triangles
RANDOMIZE
SET WINDOW 1,320,1,200
DO
    LET i=1
    DO
        LET x=INT(RND * 320)+1
        LET y=INT(RND * 200)+1
        IF i=1 THEN
           LET x0=x
           LET y0=y
        END IF
        PLOT x,y;
        PAUSE 1
        LET i=i+1
    LOOP UNTIL i>3
    PLOT x0,y0
    SET COLOR INT(RND * 6)+1
    SET BACK INT(RND * 16)
LOOP UNTIL KEY INPUT
END
```

为了画封闭三角形，第一个随机产生的点记在 X0,Y0 变量中，以便第三个点产生时，自动画出一个三角形，去消直线定向作用。该程序运行时，有关 GRAPHICS 图形模式中可选的前景色、背景色和改变，以及图形“隐藏”现像可看得更清楚。

图形窗口具有自动修剪功能，如果所画的直线始点在窗口中，而终点在窗口外，则窗口外的部分自动被修剪，只画出窗口中的直线部分，反之亦然。

8.1.5　图形文字设置

为了对照分析，人们往往喜欢将多根曲线画在同一张图上。True BASIC 允许将同一坐标的多条曲线画在同一张图上，并且允许程序在任何时刻改变窗口坐标，将不同坐标或不同单位的多条曲线显示在同一窗口中。为了清晰地表示不同的曲线，往往采用不同的彩色，或附加文字标注，或两者皆用的解决方法。

图形文字标注语句格式：

PLOT TEXT,AT <X>,<Y>:< 字符串表达式 >

其中 X,Y 为数值表达式，该语句用当前的前景色，在窗口 X,Y 坐标指定处，水平地标注字符串，如果直接使用字符串，则标注的字符串必须包括在双引号中。

在 GRAPHICS 图形模式下，一个屏幕行可以显示 40 个字符，与窗口坐标设置无关，与窗口占用的屏幕大小无关。

例 8.3　CARS 程序采用实际坐标设置缺省窗口，在屏幕上画出两条不同坐标的曲线，1900～1980 年汽车销售量统计曲线，与同一期间人口增长统计曲线。两条曲线水平坐标相同，但纵坐标不相同，因此利用文字标注。程序在黑色的背景下，用黄色先画出汽车销售量统

计曲线，并标注“CARS”，然后，改用白色画人口增长统计曲线，标注“POPULATION”。程序运行结果见图 8.1。

```
! CARS PROG.  Plot number of cars sold
! from 1900 to 1980
! Also plot population during same period
!
SET COLOR "yellow"
SET WINDOW 1900,1980,0,1e7
FOR year=1900 TO 1980 STEP 10
    READ cars
    PLOT year,cars;
NEXT year
PLOT
PLOT TEXT, AT 1930,2.8e6:"cars"
! set new window, read & plot population
! data
SET WINDOW 1900,1980,0,250e6
SET COLOR "white"
FOR year=1900 TO 1980 STEP 10
    READ pop
    PLOT year,pop;
NEXT year
PLOT
PLOT TEXT, at 1902,95e6:"population"
! cars production data
DATA 4190,187e3,2.2e6,3.3e6,4.4e6
DATA 8e6,7.8e6,8.2e6,8e6
! population data
DATA 76e6,92e6,105.7e2,122.8e6
DATA 131.7e6,150.7e6
DATA 179.3e6,203.3e6,226.5e6
END
```

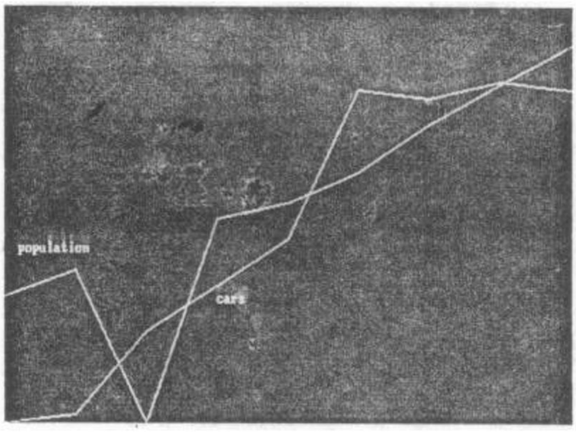


图 8.1　1900—1980 年汽车销售、人口增长统计曲线

8.1.6　图形着色

PLOT AREA：语句在窗口中画一个着色的封闭区域。语句格式为：

PLOT AREA：<X1>，<Y1>；<X2>，<Y2>；…，<Xn>，<Yn>

其中 X1，Y1，X2，Y2，…，Xn，Yn 为数值表达式，分别代表点 1，点 2，…，点 n 的坐标值。该语句首先用当前前景色画出区域的外沿线，从第一点开始画直线到第二点，再从第二个点画线到第三个点，依此类推，最后，将最后一个点和第一个点用直线连接，构成一个封闭的区域，并用前景色填满整个封闭区域。

图形封闭区域的着色也可以由程序控制。图形着色语句为：

FLOOD <X>，<Y>

该语句使用当前的前景色，给窗口中包含 X，Y 坐标点的封闭区域着色。和 PLOT AREA 语句不同，它可以使用与区域外沿线不同的彩色着色。该语句先改变封闭区域中（X，Y）坐标点的颜色，然后，用此颜色扩散到整个封闭区域。

用 FLOOD 语句着色与图形文字标注一样，必须指出封闭区域中开始着色点的坐标，有

时，难以做得十分合适，下一节介绍的图形坐标拾取语句，为人机会话方式着色，标注文字提供合适的方法。

例 8.4　BROAD 程序画出一张棋盘，利用 FLOOD 语句着色，见图 8.2。遗憾的是棋盘的每格不是正方形，在下节中将改进。

```
! board prog.
! OPEN SCREEN #1:0.25,0.75,0.10,0.86
SET WINDOW 0,8,0,8
SET COLOR "yellow"
FOR s=0 TO 8
    PLOT 0,s;8,s
    PLOT s,0;s,8
NEXT s
FOR x=0 TO 7
    FOR y=0 TO 7
        IF MOD(x+y,2)=0 THEN
           FLOOD x+.5, y+.5
        END IF
    NEXT y
NEXT x
END
```

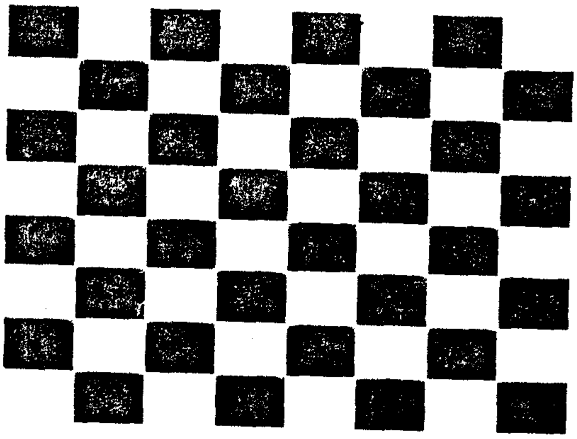


图 8.2　着色的棋盘

8.2　函数曲线图与多窗屏幕设置

8.2.1　HIRES 图形模式

在绘制函数曲线时，人们总希望能够获得一条光滑的曲线。一般情况下，曲线用直线插值、小段直线细分构成，如果系统具有多分辨率的图形模式，采用高分辨率图形模式往往能较好地满足要求。

各种计算机不仅可用的颜色差异很大，而且都有多种图形模式可供选用。参看表 8.1，IBM－PC 彩色图形卡（CGA）有两种图形模式：GRAPHICS 图形模式，HIRES 图形模式。GRAPHICS 是图形程序的缺省模式，它有 320×200 像素点阵，有 7 种可选用的前景色和 16 种可供选用的背景色，一行可以显示 40 个西文字符。HIRES 图形模式有 640×200 像素点阵，即水平方向分辨率比 GRAPHICS 模式提高 1 倍，一行可以显示 80 个西文字符，或在 CCDOS 支持下显示 40 个汉字。在 HIRES 图形模式下，可供选用前景色共有 16 种，见表 8.4，但失去了多种彩色同时使用的特性。表中色码 8～13 和 15 无色名，选用时只能使用编号数。编号 8～15 代表的颜色与编号 0～7 相同，只是它为高亮度。HIRES 模式前景色设置语句与 GRAPHICS 图形模式一样，在前景色改变时，已画图形的彩色亦跟着变，保持整个窗口图形只有一种前景色。但 SET BACK 语句失去作用，背景色只能是黑色，不允许设置。

表 8.4 HIRES 图形模式前景可选的颜色

编号	色名	说明	编号	色名	说明
0	BLACK	黑	8	/	灰
1	BLUE	兰	9	/	亮兰
2	GREEN	录	10	/	亮录
3	CYAN	青兰	11	/	亮青兰
4	RED	红	12	/	亮红
5	MAGEMTA	洋红	13	/	亮洋红
6	BROWN *	棕	14	YELLOW *	黄
7	WHITE	白	15	/	亮白

2.03 版本 True BASIC 还支持高档的图形卡(EGA)和增强彩色显示器，增加了 EGAHIRES 图形模式，EGAHIRES 模式有 640×350 像素点阵，具有 16 种可同时使用的前景彩色，背景也有一样的 16 种可选彩色。一行可以显示 80 个西文字符，或在 CCDOS 支持下显示 80 个汉字，使中、西文显示一样方便。

在需要用 HIRES 或 EGAHIRES 模式作图时，首先要用设置语句设置成相应的模式，才能作图。

例 8.5 SINE 程序设置高分辨 EGAHIRES 图形模式，在 −180°～180°区间，步距 0.5°，画正弦曲线，见图 8.3。由于分段较细，不仅曲线光滑，而且，如果存在整个 −180°～180°区间不能被步距整除的影响也可以忽略。但受分辨率限制，分段过细并不能再改善曲线光滑程度，只会增加作图时间。

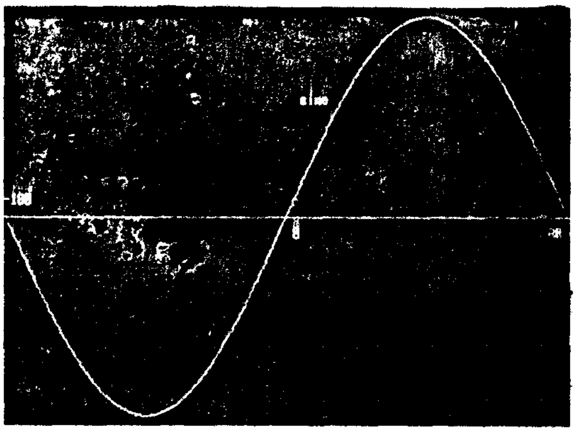


图 8.3 正弦曲线

```
! Sine prog.  plots a sine curve
SET MODE "egahires"
OPTION ANGLE DEGREES
SET WINDOW −180,180,−1,1
SET COLOR 15
PLOT −180,0;180,0
FOR x=−180 to 180 STEP .5
    PLOT LINES;x,sin(x);
NEXT x
SET CURSOR 6,43
PRINT "sine"
PLOT TEXT,AT 3,−.1 ;"0"
PLOT TEXT,AT 165,−.1;"180"
PLOT TEXT,AT −180,.05;"−180"
END
```

例 8.6 WHEEL 程序绘制一个圆形幅轮，它将单位圆的圆周作 N 等分，每个等分点依

次与其它点连成直线，构成幅轮。如果在坐标未设置的缺省窗口中，作出的将是一个椭圆幅轮。为改变图形比例，以窗口坐标为准，改变窗口使用屏幕部分，开辟一个新窗口 #1，占用屏幕 0.2，0.75，0.1，0.86 的范围。图 8.4 是 n= 20 时，程序运行的结果。

```
! wheel prog.
INPUT PROMPT "n=":n
SET MODE "egahires"
OPEN #2:SCREEN .2,.75,.1,.86
SET WINDOW -1,1,-1,1
SET COLOR 15
LET a=360/n
LET ra=a/57.29578
LET rr=1
FOR x=1 to n
    LET sy=SIN(x * ra) * rr
    LET sx=COS(x * ra) * rr
    FOR y=x to n
        LET cy=SIN(y * ra) * rr
        LET cx=COS(y * ra) * rr
        PLOT sx,sy;cx,cy
    NEXT y
NEXT x
END
```

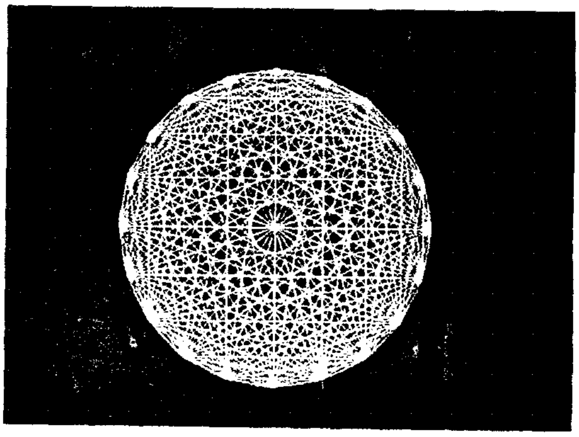


图 8.4 幅轮 (n=20)

说明：

(1)不同的显示器往往有不同的纵横比例，应事先作调整，见 8.3 节。

(2)本例已经将屏幕划分成 9 个区域，但只定义使用其中一个窗口，由图 8.4 可见，圆是由细分线段构成的。

8.2.2 MAT PLOT 语句

在很多情况下，把函数值存储在一个数组里，用一条 MAT PLOT 语句画出函数曲线，会带来很多方便。在 True BASIC 中有 3 条 MAT PLOT 语句：

MAT PLOT POINTS:<数组>	画点列
MAT PLOT LINES:<数组>	画折线或曲线
MAT PLOT ATEA:<数组>	画着色封闭区

这些语句中，<数组>必须是两个二维数值型数组，存储图形中各点的坐标值，并且第一维代表 X 坐标，第二维代表 Y 坐标。

一个 MAT PLOT 语句。只能输出一个数组的图形。

例 8.7 LISSAJ 程序利用 MAT PLOT AREA 语句画着色李沙育图，结果见图 8.5 。判断一个点在区域内部必需满足的条件是：或者它是在边界线上，或者从这一点出发，向任何方向的射线穿过区域边界线的次数为奇数。

```
! Lissaj prog. draws a lissajous figure, and
! fill it in
! DIM points(201,2)
! SET MODE "egahires"
SET COLOR 15
SET WINDOW -1,1,-1,1
FOR i=0 TO 2 * PI STEP PI/100
    LET c=c+1
    LET points(c,1)=SIN(3 * i)
    LET points(c,2)=COS(5 * i)
NEXT i
MAT PLOT AREA:points
END
```

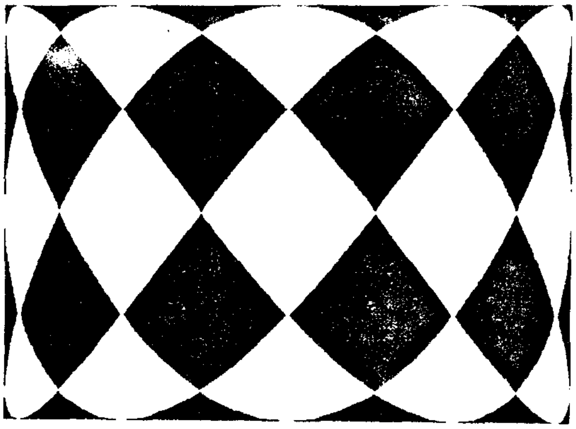


图 8·5　着色李沙育图

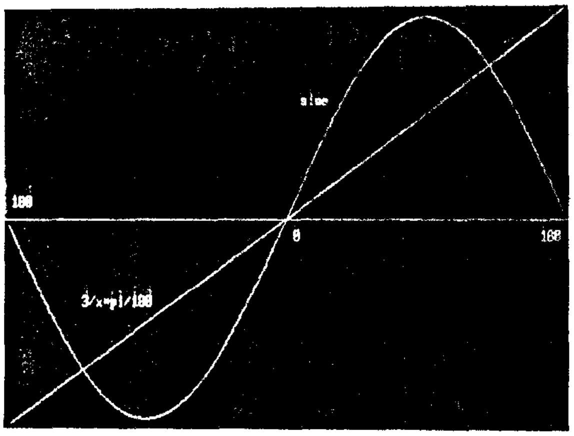


图 8·6　两条相交曲线

例 8.8　ACROSS 程序绘制 －180°～180°的两条函数曲线：X/3＊ PI/180 和 SIN(X)。由于不能边算边画，计算数据存于两个数组 S,L 中，用两个 MAT PLOT LINES 语句作图，并用 PLOT TEXT 语句标注曲线和 X 坐标。结果见图 8.6。

```
! across porg. plots two across curves
OPTION ANGLE degrees
DIM s(730,2),l(730,2)
SET MODE "egahires"
SET WINDOW -181,181,-1.05,1.05
SET COLOR 15
PLOT -181,0;181,0
LET i=1
FOR x=-180 to 180 step .5
    LET s(i,1),l(i,1)=x
    LET s(i,2)=sin(x)
    LET l(i,2)=x/3 * PI/180
    LET i=i+1
NEXT x
MAT PLOT LINES:s
MAT PLOT LINES:l
SET CURSOR 6,43
PRINT "sine"
PLOT TEXT,AT 3,-.1 :"0"
PLOT TEXT,AT 165,-.1:"180"
PLOT TEXT,AT -180,.05:"-180"
PLOT TEXT,AT -130,-.5:"3/x * PI/180"
END
```

8.2.3　窗口操作和屏幕划分

以上作图除例 8.6 外，使用的是 #0 缺省窗口，无论窗口坐标大小，缺省窗口占用了整个屏幕。可以发现，在设置的窗口纵横坐标相同，要求画一个正方形，画出的是矩形，见图 8.2。想画一个圆，但得到的是椭圆，这是由于屏幕的纵横比例不相等造成。为获得较客观、成比例的图形，要求改变窗口占用屏幕的范围。实际应用中，用户也往往希望屏幕上出现多个窗口，以显示不同图形和文本。

True BASIC 允许用户重新划分屏幕，建立一个或若干个窗口。为便于屏幕划分，整个屏幕看成一个二维直角坐标系，坐标原点位于屏幕的左下角，水平坐标轴 Xs 的正方向是自左向右，纵坐标轴 Ys 的正方向是自下而上，水平坐标、纵坐标的最小值为 0，最大值为 1。

(1)窗口打开和屏幕划分

True BASIC 将显示器看成是特殊的文件，程序在处理文件前必须先打开文件，建立文件的通道联系。窗口操作也一样，图形程序开始运行时，缺省窗口 #0 已经自动打开，它将整个显示屏幕指定为单一窗口。在屏幕重新划分时，程序可以用打开语句建立新的屏幕窗口，指定窗口使用的通道及屏幕范围。语句格式为：

OPEN #<通道>:SCREEN <Xl>,<Xr>,<Yb>,<Yt>

语句作用是打开一个新窗口，指定 #<通道>的值为窗口的通道号，指定窗口在屏幕上的位置。<通道>是数值表达式，取值范围 1～999 的整数。数字表达式 Xl、Xr、Yb、Yt 是屏幕坐标值，指明新窗口在屏幕左、右、下、上的位置，屏幕坐标水平、垂直方向的取值范围都是 0～1，与实际屏幕大小无关。实际窗口坐标仍由 SET WINDOW 语句设置，此后，与以往一样可以用实际窗口坐标画图，标注文字或着色，只是在 #<通道>相应的新窗口中输出，不再是缺省窗口。

例如，OPEN #1:SCREEN 0,0.49,0,1
SET WINDOW －10,10,－10,10

OPEN 语句打开并指定屏幕的左半侧为 #1 号窗口，语句 SET WINDOW 设定窗口 X、Y 坐标取值范围是 －10～10。

新窗口被打开后，所有的后继输出操作如 PRINT、PLOT 和 DRAW 都在该窗口进行，称当前窗口。

例 8.9　例 8.4 BOARD 程序本希望画出一张正方形方格的棋盘，但如图 8・2 所见，棋盘的方格是长方形。本程序只在程序 BOARD 中增加一条屏幕设置语句，该语句打开 #1 号窗口，并指定它占用屏幕的 0.25,0.75,0.10,0.86 部分，窗口坐标设置都未变，这时画出是真正由方格组成的棋盘，见图 8.7。

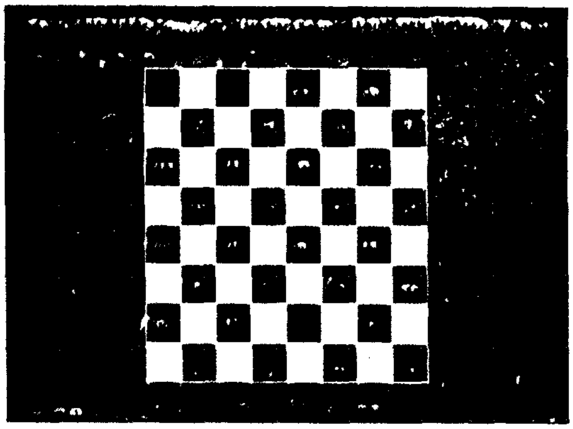


图 8・7　方格棋盘

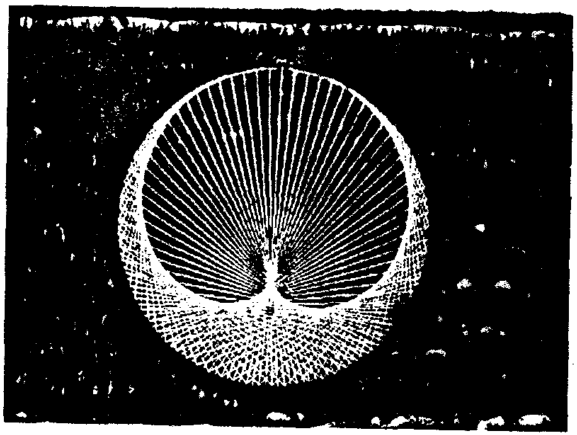


图 8・8　连线图(n＝180,d＝1)

例 8.10　connect 程序打开 #2 号窗口，并指定窗口占用屏幕 0.25,0.75,0.1,0.86 部分，窗口坐标是 －1,1,－1,1 。程序在单位圆的圆周上取 n 个等分点，每个等分点 (i) 与相间 d＊i－1 的点相连，构成连线图。在 d 取值不同时，可产生各种美丽的梅花辫，图 8.8 是 n＝180,d＝1 的连线图。

```
! Connect prog.
INPUT PROMPT "n(>60),d(1-6)=":n,d
SET MODE "egahires"
OPEN #2:SCREEN .2,.75,.1,.86
SET WINDOW -1,1,-1,1
SET COLOR "yellow"
LET a=360/(n*57.29578)
LET x,r=1
LET y=0
```

```
PLOT x,y
FOR i=1 TO n
    LET ai=i*a
    LET x=SIN(ai)*r
    LET y=COS(ai)*r
    PLOT x,y;
    LET ac=ai+(d*i-1)*a
    LET x=SIN(ac)*r
    LET y=COS(ac)*r
```

```
   PLOT x,y
NEXT i
```

```
END
```

(2)当前窗口

True BASIC 允许在同一模式的屏幕中建立若干个不同显示方式的窗口，每个窗口必须打开、建立相关连的通道，划分占用的屏幕区后，程序使用 PRINT 和 PLOT 语句在窗口中输出文本和作图。但在某一个时刻，程序只能使用一个窗口，称当前窗口，所有的屏幕输出都指定为当前窗口。程序可以改变当前窗口，控制不同时刻在不同窗口中输出。已经打开的窗口企图再次打开，True BASEC 给出“窗口已打开”的出错信息。

在多窗口屏幕划分时，要特别注意所使用计算机的窗口特性，有的计算机提供不透明窗口，有的提供透明窗口。所以在屏幕划分时，一般不希望窗口重叠和窗口嵌套。为了解当前窗口的屏幕位置，可用查询语句：

```
ASK SCREEN <Xl>,<Xr>,<Yb>,<Yt>
```

其中，数值变量 Xl、Xr 表示当前窗口占用屏幕部分的坐标左、右值，Yb、Yt 为屏幕坐标的下、上值。

窗口操作结束后，关闭窗口。关闭窗口语句为：

```
CLOSE #<通道>
```

其中 #<通道>表示通道号，该语句关闭通道号对应的窗口。窗口被关闭后不能再使用，并自动返回 #0 窗口。程序运行结束时，除 #0 窗口外，所有其它窗口全部关闭。

(3) 多窗口的窗口切换

当屏幕上打开多个窗口时，只有当前窗口是活动窗口，True BASIC 允许改变当前窗口。窗口切换语句为：

```
WINDOW #<通道>
```

语句将 #<通道>窗口指定为当前窗口，此后所有的屏幕输出，将在这个窗口中进行。

True BASIC 还记忆每个窗口的当前颜色、光标位置、窗口坐标等，当窗口切换时，自动跟着转换，继续被切换前的输出。

8.2.4 光标设置

在当前窗口中格式化输出文本时，需要控制输出，设置光标位置，用 PRINT 语句输出文本。光标设置语句格式为：

```
SET CURSOR <row>,<col>
```

<row>、<col> 为数值表达式，语句在当前窗口中设置文本输出位置，row、col 的值分别指定光标在窗口中的行、列数。行、列数最小取值为 1，最大值取决于窗口的最大可能值。企图将光标设置在窗口之外，将得到一个出错信息。

注意：在“HIRES”、“ EGAHIRES”等图形模式下，可以设置图形窗口和文本窗口，允许两种窗口共存。在图形窗口中也允许与文本窗口一样输出文本，或在键盘输入时显示光标，但一般不希望在窗形窗口中混杂文本输入输出。

光标查询语句可以查询在当前窗口中光标设置最大可能的行、列数：

```
ASK MAX CURSOR <Xr>,<Xc>
```

数值变量 <Xr>、<Xc> 返回光标最大可能行、列值。

指示文本输入、输出位置用的光标，可以打开也可以消隐。光标开关语句格式为：

```
set cursor <"on"/"off">
```

其中，on 为显示光标，off 为关闭光标。

例 8.11 WINDOWS 程序在屏幕上开设了 3 个窗口。在 EGAHIRES 图形模式下，用彩色填充的方法使窗口不透明，程序演示了 3 个不透明重叠窗口的切换及窗口中文本输出的情况。

```
! windows1 prog. illustrates using of windows
SET MODE "egahires"
OPEN #1:SCREEN 0,.5,.5,1
SET COLOR "red"
OPEN #2:SCREEN .25,.75,.3,.8
SET COLOR "green"
OPEN #3:SCREEN .45,.95,0,.5
SET COLOR "blue"
FOR w=1 TO 3
    WINDOW #w
    SET WINDOW -w,w,-w,w
    BOX AREA -w,w,-w,w
    SET COLOR 15
    BOX CIRCLE -1,1,-1,1
    PLOT TEXT ,at -w/4,0:"#" & str $(w)
    PAUSE 2
NEXT w
END
```

8.2.5 会话式图形坐标拾取

在许多情况下，人机会话的图形坐标拾取十分有用。它不仅可在用户控制下作图、测量图形坐标，而且可以人为控制在图形中着色或标注文字等。

True BASIC 提供两条图形坐标拾取语句：键盘控制、鼠标定位器控制图形坐标拾取语句。键盘控制坐标拾取语句格式为：

```
GET POINT:<X>,<Y>
```

其中 <X>,<Y> 是两个数值变量，执行该语句时，在当前窗口中显示一个十字光标（以区别文本光标），用户用键盘上的光标控制键（↑、↓、→、←为上、下、右、左移动，PGDN、PGUP、CTRL+←、CTRL+→为下、上、左、右快速移动十字光标），当光标移到被拾取点时，按回车键，十字光标所在点的窗口坐标被拾取，并将水平坐标值存储 X 变量、垂直坐标值存储 Y 变量中，返回程序。

鼠标定位器控制图形坐标拾取语句为：

```
GET MOUSE:<X>,<Y>,<STATE>
```

其中 X、Y 两个数值变量意义与前相同。数值表达式 <STATE>的值反映了鼠标定位器当前的状态：0 表示鼠标键未按下；1 表示鼠标定位器在移动；2 表示鼠标键按下；3 表示鼠标键被释放。执行该语句时，十字光标随鼠标器移动，坐标拾取、存储都由鼠标器控制。不一定所有的计算机可以配置鼠标定标器，请参看所用机器的用户手册。

例 8.12 CURSOR1 程序设置 HIRES 图形模式，利用 GET POINT 语句，在缺省窗口中。产生十字光标，画任意折线图。当十字光标移到坐标原点（屏幕左下角）时，按回车键消去直线定向作用。继续上述步骤，可作出另一条折线。程序用 CTRL+BREAK 键强迫中断，退出运行。

```
! cursor1 prog. connects the picked points
! with line
SET mode "hires"
DO
   SET color "red"
   GET POINT: x,y
    IF x=0 and y=0 then
       PLOT
   ELSE
       PLOT x,y;
    END IF                                    END
  LOOP
```

8.3　子图与子图变换

PLOT 语句组成的图形子程序绘制一个独立的简单图形，建立了一个重复使用的图形模板。与一般子程序一样，每调用一次图形子程序，恰似按模板画出一个相似的图形。此外，True BASIC 还允许用户定义子图。子图比图形子程序更灵活；子图不仅建立了重复使用的图形模板，并允许作几何变换，改变它的位置、大小、形状，或旋转，产生相似的或变换了的一族图形。利用简单的子图及其变换，可以方便地构成各种复杂图形。

8.3.1　子图

子图实质上是一个由 PICTURE / END PICTURE 结构定义的、由 DRAW 语句调用的、绘制图形用的独立子程序。调用子图的程序称主程序。内部子图在主程序中定义，外部子图是一个可独立编译的程序单位，也可以收集、存储，构成子图程序库文件。

(1)子图的定义。子图的定义结构为：

```
PICTURE <子图名>[(<形式参数表>)]
        ..................
        ……语句块……
        ..................
            PLOT
        ..................
        ……语句块……
        ..................
END PICTURE
```

子图的定义结构由 PICTURE 语句开始，由 END PICTURE 结束。

(2)子图调用语句。语句为：

DRAW <子图名>[(<实在参数表>)]

与子程序一样，子图可以不带参数，也可以有参数，有内部子图和外部子图之分。内部子图在主程序中定义，可以在主程序的首部、尾部或任意位置，不会形成陷井。主程序运行时会绕过子图定义，仅当主程序执行 DRAW 子图调用语句时，子图程序才被执行。内部子图和主程序共享变量。外部子图是在主程序外定义，或是库文件中定义的子图，它是一个程序单位，它的所有变量都是局部变量，即使外部子图定义的变量和主程序同名，也不会发生变量冲突。无论是内部子图或外部子图都可以有参数，参数是子图范围有效的局部变量。参数由主程序调用语句的对应实在参数赋值。

位于主程序 END 语句后的外部子图，主程序和内部子图一样调用。如果主程序要调用程序库中的子图，则在主程序首部，在子图调用前必须用 LIBRARY 语句声明使用的库文件：

LIBRARY <引用库文件名>

此后，主程序用 DRAW 语句调用库中的子图。

8.3.2　子图的几何变换

到目前为止，尚未看到定义子图更多的优越性，True BASEC 提供了子图几何变换能力，

使子图简单地变成灵活多变的图形模板。

True BASIC 为 DRAW 语句的使用提供了 5 种几何变换，使子图能缩小、放大、平移、旋转或剪切，见表 8.5。

表 8.5　子图几何变换

变换名	变换作用	说　明
SHIFT(a,b)	(X,Y)→(X+a,Y+b)	平移变换
SCALE(a,b)	(X,Y)→(X * a,Y * b)	比例变换
SCALE(a)	(X,Y)→(X * a,Y * a)	等比例变换
ROTATE(a)	(X,Y)→(X * cosa−Y * sina,X * sina−Y * cosa)	旋转变换，绕坐标原点逆时针转 a 角 *
SHEAR(a)	(X,Y)→(X+Y * tana,Y)	剪切变换，垂线向前（顺时针）倾斜 a 角 *

注：除非有 OPTION ANGLE DEGREES 语句出现，否则角度单位为弧度。

包含几何变换的子图调用语句为：

DRAW <子图名> [(<形参表>)] [(WITH <变换> [{ * <变换>}]

其中：　<变换>=<变换名>(<参数表>)

变换可以是表 8.5 中 5 种之一的单一变换。执行语句时，屏幕上显示作指定几何变换后的子图。当指定复合变换时，<变换>间用星号分隔，系统规定变换从左而右进行，屏幕显示最后的变换结果图形。

例 8.13　SQUARES 程序定义了带参数的外部子图 SQUARE，它按参数值作边长画一个正方形。在主程序的循环中，利用循环变量作实参，每次调用作 PI/40 弧度的旋转变换，构成一族大小变化、旋转的正方形，如图 8.9 所示。

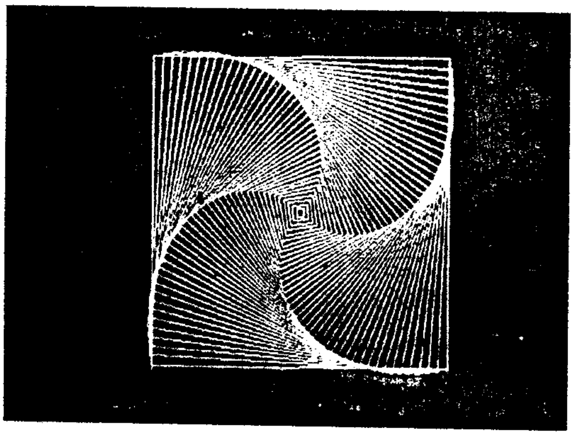


图 8.9　一族旋转嵌套的正方形

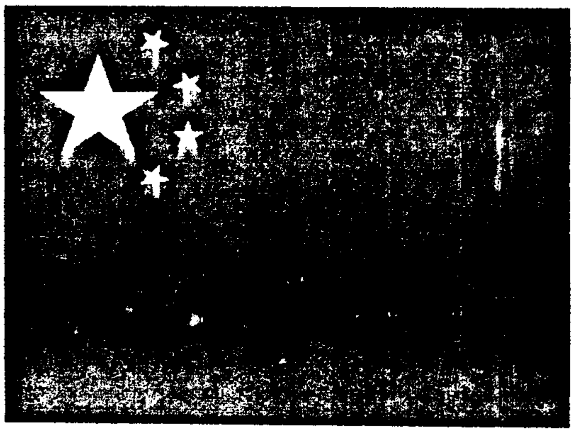


图 8·10　五星红旗

```
! squares prog.
SET MODE "egahires"
OPEN #1:SCREEN .2,.75,.1,.85
SET WINDOW -15,15,-15,15
FOR i=.5 TO 12 STEP .5
  DRAW square(i) WITH ROTATE(i * PI/40)
NEXT i
END
PICTURE square(size)
  PLOT -size,size;
  PLOT size,size;
  PLOT size,-size;
PLOT -size,-size;
PLOT -size,size
END PICTURE
```

例 8.14　CFLAG 程序绘制一面五星红旗，它定义了画一个五角星并用黄色填充的外部子图 FASTAR，星的大小及位置由参数决定。程序首先将窗口用红色填充，制成红旗，利用参数在红旗的左上角画一个大五角星，再改变参数，转动 PI/8、平移后，重复画出 4 个围绕大五角星的小星星，此时屏幕上出现一面彩色艳丽的五星红旗，见图 8.10。

```
! Cflag prog. draws a flag of five stars
SET MODE "egahires"
SET WINDOW 0,12,0,10
SET COLOR "red"
BOX AREA 0,12,0,10              ! red flag
LET p=PI/5
SET COLOR "yellow"
DRAW fastar(2,7.5,1.3)          ! big yellow star
FOR i=1 to 4                    ! five small stars
    LET x=2 * COS(p * i-PI/2)+2
    LET y=2 * SIN(p * i-PI/2)+7.5
    DRAW fastar(0,0,.3) WITH ROTATE(PI/8 * (i-2)) * SHIFT(x,y)
NEXT i
END
PICTURE fastar(a,b,r)           ! draw star
    DIM x(0:4),y(0:4)
    LET p=2 * PI/5
    FOR i=0 to 4
        LET x(i)=r * COS(MOD((i * 2),5) * p+PI/2)+a
        LET y(i)=r * SIN(MOD((i * 2),5) * p+PI/2)+b
        PLOT x(i),y(i);
    NEXT i
    PLOT x(0),y(0)              ! flood with yellow
    FLOOD a,b
    FOR i=0 TO 4
        FLOOD .5 * (x(i)-a)+a,.5 * (y(i)-b)+b
    NEXT i
    PLOT
    SET COLOR "yellow"
END PICTURE
```

8.4　BOX 语句与动画

窗口中矩区称“盒子”(BOX)。BOX 语句与当前窗口中的某个“盒子”一起工作，例如，画“盒子”的边线框，或用前景色填充画实心盒子，画盒子的内切椭圆，BOX 语句还允许将盒子图形存储，移动位置重新画出，速度快到足以产生动画的效果。

8.4.1 BOX 语句

以下是 BOX 系列语句及其作用：

BOX LINES <X1>，<X2>，<Y1>，<Y2>	画盒子边框
BOX AREA <X1>，<X2>，<Y1>，<Y2>	画实心盒子
BOX ELLIPSE <X1>，<X2>，<Y1>，<Y2>	画盒子内切椭圆
BOX CLEAR <X1>，<X2>，<Y1>，<Y2>	清除盒子中的图形
BOX KEEP <X1>，<X2>，<Y1>，<Y2>	盒子中图形保存在 K $ 中
BOX SHOW K $ AT <X>，<Y>	在 X，Y 处重现保存图形

其中 X1、Y1、X2、Y2 为数值表达式，代表盒子左下角与右上角的纵、横坐标的窗口坐标值。K $ 为盒中图形存储变量；X，Y 是盒子重现定位坐标值。BOX LINES 画盒子边框比用 PLOT 语句画矩形效率高，速度快，BOX AREA 语句比 PLOT AREA 快，并可用作给窗口中某个矩形着色。BOX ELLIPSE 语句画出刚好填满盒子的椭圆，如果盒子是一个正方形，则可以画出一个圆。它可用作屏幕纵横比调整的根据。BOX ELLIPSE 语句也可以用 BOX CIRCLE 代替。BOX 语句不受几何变换影响。

8.4.2 动画

BOX KEEP、BOX CLEAR、BOX SHOW 可以用作动画制作。BOX KEEP 语句将窗口盒子中的图形拾取，并保存在字符串变量中。BOX CLEAR 是清除盒子中的内容，并用当前背景色填充。BOX SHOW 语句将保存在字符串变量中的图形，在指定位置重新显示，其中 AT 短语指示盒子左下角的窗口坐标。如果将盒子图形存储，在某个位置显示后清除，又在下一时刻移动盒子再显示，依此重复，就能产生移动的图形。用户可以修改被存储的图形，修改与移动相结合，会产生更好的动画效果。

考虑到活动图形的背景，产生更逼真的动画效果，BOX SHOW 语句有三种扩展形式：

BOX SHOW <K $> AT <X>，<Y> USING <″and″>

BOX SHOW <K $> AT <X>，<Y> USING <″or″>

BOX SHOW <K $> AT <X>，<Y> USING <″xor″>

其中“and”、“or”、“xor”必须是字符串常数，大小写等效。它们表示存储的图形与窗口中背景图形两者不同的组合逻辑，图形像素是按位进行组合，每位只有 0 或 1 两种可能值。“and”、“or”和“xor”和同名逻辑运算一样工作。使用“and”表示，当仅当存储图形和背景图形对应的位都为 1 时，显示图形对应位才为 1，否则为 0。如果使用“OR”，则被存储图形和背景图形对应位中有一个为 1 时，图形对应位为 1。如果使用异或操作“XOR”，当存储图形和背景图形对应位值相异时，新图形相应位为 1，相同时，为 0。

对单色显示器，计算机仅用 1 位表示图形像素的明暗，亮时为 1，暗时为 0；彩色显示器用多位组合表示一个像素不同的彩色，例如，16 色像素，采用四位组合，见表 8.6。

像素按位逻辑操作可以获得不同的彩色效果，例如棕、红两色不同的组合逻辑：

棕 0110 AND 红 0100，结果为：0100 红；

棕 0110 OR 红 0100，结果为：0110 棕；

棕 0110 XOR 红 0100，结果为：0010 绿。

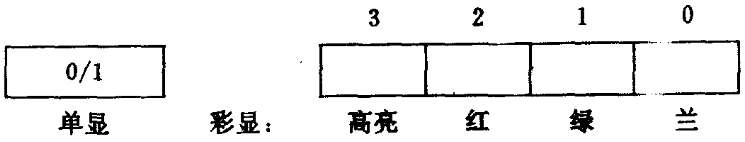


表 8.6　一个 16 色像素的编码

编　码	色　名	编　码	色　名
0000	黑	1000	灰
0001	兰	1001	亮兰
0010	绿	1010	亮绿
0011	青兰	1011	亮青兰
0100	红	1100	亮红
0101	洋红	1101	亮洋红
0110	棕	1110	黄
0111	白	1111	亮白

BOX SHOW 更进一步推广，允许存储图形和屏幕图形有 16 种可能组合：

BOX SHOW <K＄> AT <X>,<Y> USING <数值表达式>

其中<数值表达式>的值为组合代码，True BASIC 自动取整，然后作模 16 运算，得到 0 到 15 间的一个数值，每个数值都代表存储图形和屏幕图形相应位的组合方式，见表 8.7。

表 8.7　像素的 16 种组合方式

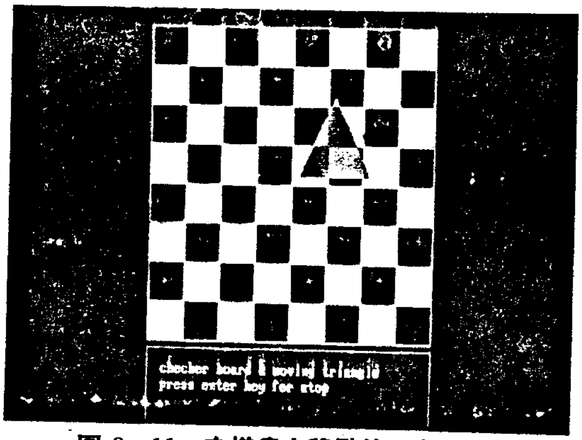
组合代码	逻辑符＃	逻辑表达式	组合代码	逻辑符＃	逻辑表达式
0	/	0	1	AND	$A\cdot B$
2	/	$A\cdot\bar{B}$	3	/	A
4	/	$\bar{A}\cdot B$	5	/	B
6	XOR	$A\cdot\bar{B}+\bar{A}\cdot B$	7	OR	$A+B$
8	/	$\bar{A}\cdot\bar{B}$	9	/	$A\cdot B+\bar{A}\cdot\bar{B}$
10	/	$\bar{B}$	11	/	$A+\bar{B}$
12	/	$\bar{A}$	13	/	$\bar{A}+B$
14	/	$\bar{A}+\bar{B}$	15	/	1

注：A 代表存储图组合像素位串，B 代表屏幕图的位串；

组合逻辑符是第一类 BOX SHOW 扩展表示中的字符串

说明：BOX SHOW 的扩展形式并非所有计算机都能使用，取决于图像存储方式，不同的计算机可能有完全不同的结果。

例 8.15　MOVE 程序开设两个窗口，一个作文本输出，一个作图形输出。在图形窗中首先画出黄色填充的三角形，并将它存储于 K＄变量中，在文本窗口中说明它就是可移动的三角形。在图形窗口清除后，画方格棋盘，并用红色填充，它作为背景图形。让三角形从棋盘左下角开始，按对角线来回移动，得到一个活动的三角形，在文本窗口中说明棋盘和移动的三角形。程序中三角形和棋盘的显示组合，采用“XOR”，在某个时刻，屏幕显示如图 8.11 所示。



```
checker board & moving triangle
press enter key for stop
```

图 8・11　在棋盘上移动的三角形

```
! (move prog. )
! moveing trianle on checker board
SET MODE "egahires"
OPEN #1;SCREEN .25,.75,.21,.96
SET WINDOW 0,8,0,8
SET COLOR "yellow"
OPEN #2;SCREEN .25,.75,0,.19
SET WINDOW 0,10,0,10
SET COLOR "green"
BOX LINES 0,10,0,10
SET CURSOR 2,3
PRINT "triangle"
WINDOW #1
DRAW TRIANGLE WITH SHIFT(4,4)
BOX KEEP 4,6,4,6 in k1 $
PAUSE 1
CLEAR
WINDOW #2
BOX CLEAR 0.1,9.9,0.1,9.9
SET CURSOR 1,3
PRINT "checker board & moving triangle"
WINDOW #1
SET COLOR "red"
DRAW board
LET a=8
LET b=0
LET delta=-1
LET step=.1
DO
   LET c=a
   LET a=b
   LET b=c
   LET delta=-delta
   FOR x1=a to b step delta * step
       LET x2=x1+2
       LET y1=x1
       LET y2=x2
       BOX KEEP x1,x2,y1,y2+.1 in k $
       BOX SHOW k1 $ at x1,y1 using "xor"
       PAUSE .1
       BOX SHOW k $ at x1,y1
   NEXT x1
LOOP UNTIL KEY INPUT
END
PICTURE triangle
    PLOT AREA : 0,0;2,0;1,2
END PICTURE
PICTURE BOARD
    FOR s=0 to 8
        PLOT 0,s;8,s
        PLOT s,0;s,8
    NEXT s
    FOR x=0 to 7
        FOR y=0 to 7
            IF mod(x+y,2)=0 then
               FLOOD x+.5,y+.5
            END IF
        NEXT y
    NEXT x
END PICTURE
```

8.5 声音

True BASEC 提供了两种在计算机上的发声语句:PLAY,SOUND。PLAY 语句可用类似音乐标准语言的术语描述乐曲,演奏出明快动听的音乐。SOUND 语句,按指定的要求发出不同频率和持续时间的声音。

8.5.1 PALY 语句

PLAY 语句可以在计算机上演奏简单的音乐。语句格式为:

PLAY [<字符串表达式>]

其中:字符串表达式是由类似音乐语言术语描述的部分或一个乐曲,PLAY 语句将按照字符串指定的乐曲和曲调奏音乐。当字符串表达式缺省时,将在下一个语句之前等待乐曲终结,也可以作为下一个乐曲的开始,当执行其它语句时,用作伴音。

在 True BASIC 中有以下一些音乐语言术语描述字符。

(1) 音阶和音符:如同钢琴,共有 7 个音阶,开始于 C 结束于 B,对应编号为 0～6,缺省时为第 4 个音阶,音阶按照乐曲设置。

On　设置数值 n(0～6)为当前音阶,设置的音阶一直保持,直到下次音阶设置,或程序运行完毕。每个音阶有 8 个音符 A～G。

>和<　向上 (>) 或向下 (<) 移动一个音阶。如果用第 4 音阶演奏一个乐曲的音符,在">"后的音符,则将用第 5 音阶演奏。不得在音阶 0 向下或音阶 6 向上移动,否则出错。

A～G (带任选项 #,+ 或 -)　当前音阶演奏的音符,当音符后跟"#"或"+"时,音符升半音,"-"降半音。只有当钢琴对应键有黑键才允许跟任选项,否则音符无效。

Rn、Pn　休止符,n 为休止(暂停)音长。

(2)音长和附点:乐曲音符的发音长度称音长。音符后面的数值 n 是音长,音长与数字 n 成反比,也就是说,n=1 是全音符,n=2 是半音符,n=3 是三分音符,n=4 是四分音符等,以此类推。调整速度将改变所有音符的音长。音符后面的点"."为附点,表示该音符的音长将乘以 3/2,一个音符后面可有多个附点。例如,A. 将演奏 3/2 音长,而 A.. 演奏 9/4 音长,A2. 将演奏 3/4 音长等等。休止符也可加附点以延长休止时间。

Ln　设置随后音符的音长,缺省是四分音符。

Tn　设置乐曲的演奏速度。数值 n 表示一分钟演奏四分音符的个数。缺省时,为 120,这是节拍器的标准速度。

常用音速与分钟节拍数

音速	缓慢	慢	稍慢	进行速	中速	迅速	快速
拍/分	40—60	60—66	66—76	76—108	108—120	120—168	168—208

(3)演奏方式和曲调

MF　使后面的音乐在"前景"中演奏。也就是说,在演奏过程中,不执行其它语句,直到演奏结束。缺省时,为 MF 演奏方式。

MB　与 MF 相对应,MB 使后面的音乐在"后景"中演奏。也就是说,把音乐字符串的

音符放在一个缓冲区中，在音乐演奏的同时，程序可继续执行其它语句。不过，缓冲区每次最多允许存放 32 个音符(包括休止符)，自动忽略多余的音符。当程序运行结束时，后台乐曲自动停止。如果在程序结束前，要停止后台乐曲，须用带 MF 的 PLAY 语句。

MN　“正常”地演奏乐曲，每个音符按指定音长的 7/8 演奏，即在每个音符后留一点间断(1/8 音长)，使乐曲有一种干脆利落的感觉。

ML　“连奏”地演奏乐曲，每个音符按指定的音长演奏，乐曲声音显得较慢，连绵。

MS　“断奏”地演奏乐曲，每个音符仅演奏音长的 3/4，乐曲声音显得很活泼。

MN,ML,MS 是相对应的，缺省时按 MN 曲调演奏。

在音乐描述字符串中，可在任何地方插入空格，不会影响乐曲。除上述所列出的符号外，其它符号不得在音乐串中出现。

例 8.16　SEEOFF 程序按 ML、T130 曲调先演奏一遍乐曲“送别”，随后改用 MS、T200 曲调演奏第二遍，可以听出两次演奏的曲调有明显的差别。

```
! PLAY "SEE OFF"
LET A$="T130 ML O4 G4 E8 G8 >C2"
FOR I=1 TO 2
    PLAY A$
    PLAY "O4 A4 >C4 <G2 P64 G4 C8 D8 E4 D8 C8 D2"
    PLAY "G4 E8 G8 >C4 C8 <B8"
    PLAY "A4 >C4 <G2 P64 G4 C8 D8 F4 F8 <B8 >C2"
    PLAY "A4 >C4 P64 C2 <B4 A8 B8 >C4"
    PLAY "<A8 B8 >C8 <A8 A8 G8 E8 D8 C8 D8 D2"
    PLAY "G5 E8 G8 >C4 <B4 A4 >C4 <G2 P64 G4 C8 D8 F4 F8 <B8 >C2"
    LET A$(1:7)="T200 MS"
NEXT I
END
```

例 8.17　PIANO 程序用键盘模拟钢琴键，用户可在键盘上演奏乐曲。程序首先在显示器缺省窗口中画出钢琴键，并标注对应键盘的键符，按钢琴键定义相应键的音阶和音符。最后，提示用户可以利用计算机键盘进行随意演奏，直到按 ESC 键结束程序运行。

```
! simulate piano p22
DIM s$(0:15,2),t$(0:13,2)
FOR i=0 TO 15
    READ s$(i,1),s$(i,2)
    IF i<=13 THEN READ t$(i,1),t$(i,2)
NEXT i
SET MODE "egahires"
SET WINDOW 1,150,1,20
FOR i=12 TO 135 STEP 8
    SET COLOR 6
    BOX AREA i,i+7,10,15
    SET COLOR 15
    PLOT TEXT,AT i+3,11:s$((i-12)/8,1)
```

```
NEXT i
FOR i=24 TO 120 STEP 8
    IF i<>40 AND i<>72 AND i<>96 THEN
       SET COLOR 2
       BOX AREA i,i+7,12,15
       PLOT TEXT,AT i+3,13;t$ (i/8-2,1)
    END IF
NEXT i
PLOT TEXT,AT 100,18;"LISTEN"
DO
    FOR i=1 TO 15
        PLOT TEXT,AT (i-1)*8+23,8;CHR$ (14)
        PLAY "MN" & s$ (i,2)
        SET COLOR 2
        PLOT TEXT,AT (i-1)*8+23,8;“  ”
    NEXT i
    PLOT TEXT,AT 100,18;"          "
    PLOT TEXT,AT 24,4;"onece more ? (Y/N)"
    GET KEY a
    LET a$ =CHR$ (a)
    PLOT TEXT,AT 24,4;" "
LOOP UNTIL UCASE$ (a$ )<>"Y"
PLOT TEXT,AT 50,4;"Now play, or strike (ESC) to EXIT"
DO
    GET KEY v
    IF v=27 THEN EXIT DO
    PLAY "T150 ML"
    SELECT CASE v
          CASE 9
               PLAY "O3B"
          CASE 8
               PLAY "O5G#"
          CASE 327
               PLAY "O6A"
          CASE 328
               PLAY "O6B"
          CASE 329
               PLAY "O6C"
          CASE ELSE
               LET s1$ =CHR$ (v)
               FOR i=0 TO 15
                       IF ucase$ (s1$ )=s$ (i,1) THEN
                           PLAY s$ (i,2)
```

```
                    EXIT FOR
                END IF
                IF i<=13 AND s1$=t$(i,1) THEN
                   PLAY t$(i,2)
                   EX IT FOR
               END IF
          NEXT i
   END SELECT
LOOP
DATA TB,O3B,1," ",Q,O4C,2,O4C#,W,O4D,3,O4D#,E,O4E,4," ",R,O4F,5,O4F#
DATA T,O4G,6,O4G#,Y,O5A,7,O5A#,U,O5B,8," ",I,O5C,9,O5C#,O,O5D,0,O5D#
DATA P,O5E,-," ",[,O5F,=,O5F#,],O5G,BS,O5G#,HM,O6A,/,O6A#
DATA UA,O6B,PU,O6C
END
```

8.5.2 发音语句

SOUND 语句可使计算机发出某个频率和一定持续时间的声音，语句格式为：

SOUND <频率>,<持续时间>

其中，频率以赫兹(Hz)为单位，持续时间以秒(s)为单位，频率的取值范围是 27～32767 之间的数字表达式，持续时间是在 0～65535 之间的数字表达式。

SOUND 语句按给定频率在规定时间内发出响声。当语句被执行时，声音即开始，在发声的同时，程序继续往下执行，直到持续的时间完结，或者另一个持续时间为 0 的 SORND 语句被执行或程序结束。如果后面一个 SOUND 语句的持续时间不为 0，则必须等到前一个 SOUND 语句结束，然后才开始后一个 SOUND 语句的发声。

以下程序模仿一些熟悉而有趣的声音。

例 8.18　STREN 程序模仿警笛声，它由 400Hz～815Hz 共 15 种频率，每种频率发声 0.2 秒，自低而高上升，再自高而低下降，共重复 10 次所发的声音。

```
! STREN prog.
FOR cycle=1 to 10
   IF cycle/2<>INT(cycle/2) THEN
      LET a=400
      LET b=815
      LET c=15
   ELSE
      LET a=815
      LET b=400
      LET c=-15
   END IF
   FOR freq=a to b STEP c
      SOUND freq,.02
   NEXT freq
NEXT cycle
END
```

例 8.19　PHONE 模仿电话电子蜂鸣器的响声，程序使用 600Hz 及 1600Hz 音频交替发声，每种频率持续 0.03 秒，共重复 8 次。

```
! PHONE prog.
FOR ring=1 to 10
   FOR i=1 to 30
      SOUND 600,.03
      SOUND 1500,.03
   NEXT i
   SOUND 32767,2
NEXT ring
END
```

小　结

（1）配备有彩色图形卡的计算机具有图形功能，IBM－PC 计算机有 GRAPHICS 和 HIRES 、EGAHIRES 高分辨率图形模式。GRAPHICS 模式是图形程序的缺省模式。程序可以用 SET MODE 语句设置所需的模式。

（2）用户在显示器的窗口中作图，计算机将图形窗口看作特殊的文件。＃0 为缺省图形窗口，它占据整个屏幕，并且总是打开的。程序可指定屏幕的一部分，建立、打开新窗口，也可以建立不同通道号的多个窗口，它们各占有屏幕的不同部分。只有当前窗口是图形、文字输出的目标，在屏幕上只有一个窗口时，窗口打开后，它是唯一的活动窗口。在有多个窗口时，WINDOW 语句用作改变当前窗口。

（3）图形窗口采用实际坐标，SET WINDOW 语句用来改变窗口坐标，用户使用实际坐标在窗口中作图。

（4）子图由 DRAW 语句调用，由 PICTURE/END PICTURE 语句定义的。它与子程序类似，定义一次可以反复使用，它也有内、外子图之分，也可以带有参数，单独编译建立子图库。不同的是子图调用可以进行图形几何变换，使用更灵活，应用更广。

True BASIC 的子图几何变换有平移、缩放、旋转和剪切等 5 种，允许单变换，也允许作多种变换，复合变换。

（5）True BASEC 提供了两种在计算机上发声语句：PLAY，SOUND。PLAY 语句可用类似音乐标准语言的术语描述乐曲，演奏出明快动听的音乐。SOUND 语句，按指定的要求发出不同频率和持续时间的声音。

习　题

1. 画缺省窗口的边框，并在窗口中间画一个等腰三角形，不允许被修剪。

2. 画边长分别是 10，8，6，4，2 的 5 个嵌套的正方形。

3. 将上题画出的 5 个正方形，在 10－8，6－4 和边长为 2 的正方形分别用绿、红、棕色填充。

4. 用高分辨图形模式、红色前景色画 $-\pi \sim \pi$ 的余弦函数曲线。

5. 要求同上，画正切函数曲线。

6. 画 $-\pi \sim \pi$ 范围的正弦和余弦函数曲线，用图形输入方式标出坐标，原点和曲线标注。

7. 在屏幕上左右对半开设两个窗口，左窗口用作作图，右窗口用作图形文字注解，要求在左窗口作出 $-\pi \sim \pi$ 的正弦函数曲线，而右半窗口中显 SINE 字样。

8. 编写画五边形的子图形程序，调用子图并带 6°角转动变换，画出 10 个嵌套旋转的五边形族。

9. 编写画五边形的子图形程序。

第9章　结构化程序设计方法

经过前面几章的学习，我们已经掌握了True BASIC语言的基本知识，读过、写过不少有关程序，对程序设计过程也有了一定的认识和了解，但是还不能说程序设计工作就是我们的“拿手好戏”了。我们还应该从整体上纵观程序设计，继续学习、了解程序设计的方法。设计一个较大型程序系统，设计人员不仅需要多方面的理论知识，还需要有设计程序的实践经验。对于一个多人合作的任务，更应有配合协作的奉献精神。可以说，程序设计是一项十分复杂的系统工程。

本章从程序和程序设计的概念出发，首先介绍开发程序系统的全过程，然后分析程序系统中的错误类型，用调试、测试手段去排除程序中的故障，最后着重讨论了结构化程序设计的理论和方法。学习本章的目的是使读者能从全局的观点上更好地驾驭程序与程序设计方法，更快地提高程序设计的水平。

9.1　程序与程序设计

9.1.1　程序设计的概念

随着计算机技术的高度发展和计算机应用的不断深入，在程序设计上人们花费了大量的人力和物力，所付出的劳动量与程序成指数曲线急剧上升。更值得注意的是，人们仅满足于编写一个可执行程序，这种程序往往质量低劣，价格昂贵、运行起来可靠性差，甚至错误百出。这就是60年代“软件危机”的局面。人们希望在理论上和实践上从科学的角度总结程序和程序设计的规律，提高程序设计效率，解决“软件危机”问题。

1969年荷兰学者E. W. Dijkstra等人面对“软件危机”，提出了结构化程序设计(Structured Programming)的理论和方法，它是处理复杂任务并能获得清晰程序结构的基本方法。1976年瑞士学者N. With指出要从算法和数据结构两个方面统一认识程序和程序设计的本质，他写下了著名的言简意赅的公式：

程序＝算法＋数据结构

程序设计包括结构设计和程序编写两个过程，即程序的结构设计和程序编码的全过程，也可以用下列公式来描述程序设计：

程序设计＝(算法＋数据结构＋结构化程序设计方法)＋语言工具

事实上，程序设计是算法、数据结构和程序设计方法三者的统一。算法是程序的核心，采用计算机语言来描写算法；数据结构是程序数据的组织和构造方法，它表达了数据和数据间的关系；结构化程序设计方法是程序设计的技术措施，是设计优质、高效程序的可靠保障。

上面所描述的程序和程序设计的两个公式对初学程序设计的人来说不可能理解得很深刻。我们也可以从另一个角度来讨论什么是程序，什么是程序设计。程序是有序的计算机指令集合，指令集合是一条一条编码，它是计算机CPU能够执行的操作命令。程序设计是编写

计算机程序的方法和过程。这就是说，要用计算机解决实际问题，需要确定解决问题的方法和过程，这种方法还得用机器指令或高级语言描述出来，这个过程就是程序设计的过程。

程序设计工作是每一个从事于计算机系统理论研究和计算机实际应用人员的最重要的工作。程序设计工作可以训练程序员的素质。一台计算机的功能和价格，除了看硬件的配置外，还在很大程度上取决于软件系统功能和质量，所以进行良好的程序设计是充分发挥计算机系统各种功能的重要途径。

9.1.2 评价程序系统的标准

70年代，一个占用存储空间少、运行速度快的程序可以算得上是个好程序。先请看下面的简例：

例 9.1 有数组 a(m,m)，b(m,m)，将 b(i,j)的元素分别送到 a(i,j)的对应单元中。其中，i,j=1,2,…,m。

程序 1：

```
...
FOR j=1 TO m
    FOR i=1 TO m
        LET a(i,j)=b(i,j)
    NEXT i
NEXT j
...
```

程序 2：

```
...
FOR i= 1 TO m
    FOR j = 1 TO m
        LET a(i,j) = b(i,j)
    NEXT j
NEXT i
...
```

如果数组元素是“按列排列”，那末最前边的下标变化最快，所以在循环嵌套时应使它对应最内层的循环控制变量，从而提高运算速度。根据这个规则，程序 1 的运行速度比程序 2 的运行速度快得多，因为程序 1 是按以下语句来执行的：

```
...
FOR j = 1 TO m
    LET k = m * (j-1)
    FOR i = 1 TO m
        LET a(i+k) = b(i+k)
    NEXT i
NEXT j
...
```

这里运用了二维数组换算公式，省去了不少 k=m * (j-1)的重复计算，从而提高了计算机运行速度。前面假定了数组元素在计算机内是按列排列的，程序 1 速度比程序 2 要快。反之，如果数组元素是“按行排列”，程序 1 运行速度就会比程序 2 要慢。

当前，大规模集成电路的应用使计算机存储空间不断增大，运算速度不断加快。因此，要求程序运行时间短，节省内存已不是主要矛盾。一个好的程序系统应该是设计合理，价格便宜，易读易维护，可靠性高的系统。当然，一个程序系统的好坏是相对的，我们只能从最主要的几个方面进行讨论。

(1)程序正确性。一个程序系统能在计算机上顺利运行通过并取得正确的结果，这个系统才有真实的价值。因此程序的正确性是衡量程序质量最重要的标准。

程序正确性的理论研究曾风靡一时，它提出了结构化程序设计的完整理论，对程序设计方法和程序设计语言都产生了巨大的影响。程序正确性是通过算法的精心设计，结构层次化、精细化，程序的仔细调试、测试这种反复的过程来实现的。要达到程序系统完全正确是一

件难事，“智者千虑，必有一失”，何况一个大的复杂的程序系统呢。一个由几个部分组成的程序，如果第 i 个部分正确率为P_i，则整个程序系统的正确概率应为$\prod_{i=1}^{n} P_i$。由此可见，一个大型程序系统要保证它的正确性，的确是件不容易的事。尽管如此，程序设计人员还是要在程序设计技术和方法上下功夫。程序的正确性仍是我们追求的最主要目标。

(2)程序结构清晰。现在已广泛采用结构化程序设计语言来设计程序，引进了结构化、模块化的方法，设计出来的程序可以达到层次分明、结构清晰。结构化程序设计方法可以把总的设计任务分成若干个子任务，然后根据子任务编写子程序，并将这诸多子程序装配在一起形成一个程序系统。结构化、模块化程序设计方法，加快了程序编写速度，提高了程序的质量。

(3)易读性、易维护性。模块化结构程序设计具有易读、易理解、易维护特性。一个程序在编写时应该做到模块结构分明，指令编写简洁，适当地选择变量，并加上必要的注释说明。这个程序系统使人容易读懂，一个易读的程序自然就是一个易理解、易维护的程序。一个程序系统随着运行时间的推移，一是一些潜伏的错误会逐步暴露出来，二是程序功能要求的增加，硬件的变更，都需要根据实际情况修改程序或更换相应的程序模块来完成程序系统的维护任务。这种修改和维护对程序系统应该是毫无影响的。

(4)程序的有效性和适应性。一个程序系统占用存储空间小、运行速度快当然是一件好事，此外，程序还应该做到适应性强、移植性好。用 True BASIC 等结构化程序设计语言编写的程序，不依赖于具体的机器，源程序可以在各类不同的机器上运行。另外，编码时注意结构化程序设计方法的应用，程序系统可以达到有效性强和适应广的要求。

(5)程序的可靠性。前面我们已经提到，虽然要保证程序正确性很难，但程序设计人员应该努力提高程序的正确率。程序的可靠性是程序系统的基本要求，程序的可靠性是指程序系统的重复性能好，关键程序运行时绝对不会出现意想不到的破坏性错误，保证程序系统的安全运行。顺便说一下，只有一个水平高的程序员才能写出好的程序来。有人曾做过这样一个试验，选择两个题目，找 12 个都是有经验的程序员(确实认为是有经验的)来编写和调试程序，在 4 个方面(编码时间 16 : 1；程序长度 6 : 1；调试时间 28 : 1；程序速度 6 : 1)进行比较最差的和最好的程序员，结果差别很大，这点很能说明问题。

由此可见，要得到一个高效且可靠的程序，关键是培养好一支软件队伍，让他们尽快地掌握科学的程序设计方法，去开拓计算机不同的应用领域。

9.1.3 用先进的程序设计理论指导程序设计

计算机基本知识、基本技能的教育是基础教育，而程序设计课程又是计算机教育和计算机应用的基础，人们应该用先进的程序设计理论指导程序设计工作。

(1)计算机语言和程序设计的关系

众所周知，从 70 年代以来，我国不少高等学校无论是计算机专业还是非计算机专业，都先后开设了计算机语言课，如 BASIC 语言。BASIC 语言简单易学，使用方便灵活，具有人—机对话交互能力，对普及计算机教育和学习程序设计起了重要的作用。但是，当时的基本 BASIC 缺乏结构化语句，人们很难用这样的 BASIC 编写出质量高的结构化、模块化的应用程序来；又由于初学者对结构化程序设计方法学(Structured Programming Methodology)了解不多，他们往往喜欢用 GOTO 语句来实现无条件转移。这样编写的程序走向紊乱，结构缠绕不

清，难于阅读和维护，就是一个有经验的软件人员也会望而生畏。

人们对程序设计进行了不断的探索和实践，逐步认识到计算机语言仅仅是程序设计的工具，不是程序设计的全部内容，学习它的目的是使用这种工具来进行程序设计。现在的程序设计课程都是用某一种高级语言为工具，以程序设计的理论和方法为基础而开设的一门计算机先导课程。

计算机语言种类繁多，除汇编语言外，还有各种通用、专用高级语言。学生在校学习一二种程序设计语言就够了，若今后需要学习其他语言，这就可以起到了举一反三、触类旁通的作用。时代在发展，计算机技术在发展，计算机语言功能在扩充，版本在更新，新的语言不断产生。那么什么语言是“程序设计语言”的最佳选择呢？可以这样认为，只要能提高软件设计效率，保证程序的可靠性和可维护性，能完成结构化、模块化程序设计任务的语言就是当前程序设计的最佳语言。True BASIC 是典型的结构化程序设计语言，它舍弃了基本 BASIC 中的 GOTO、GOSUB、ON－GOTO 等转向语句，取消了行号，充实了循环结构和判断结构的内容，引入了结构化程序设计基本思想和理论，它用全新的现代版本取代了陈旧的基本 BASIC 版本。所以 BASIC 的创始人 John G. Kemeny 和 Thomas E. Kurtz 宣称：“True BASIC 的成功开拓了 BASIC 的新纪元。”

(2)程序设计方法概述

带箭头的流程图根据程序执行先后次序可将流程箭头不加限制地转向任一程序框图的入口，这对设计简单的程序看起来方便直观，画图自由灵活。但是，用这种方法设计稍微复杂的程序，会使流程无规则地转来转去，流程图变得象乱麻一样，理不出头绪来。按流程图写出的程序自然是一个叫人难以阅读和难以理解的程序。

结构化程序设计思想要求人们不要随心所欲地去编写程序，不要挖空心思摆弄技巧，应该按照一定的合理的结构形式来设计程序。1966 年，C. Bohm 和 G. Jacopini 证明了只需三种基本结构：顺序结构、分支选择结构和循环结构，就可以实现任何结构化的程序设计，这为结构化程序设计奠定了理论基础。结构化程序一定是由上述三种基本结构所组成，不能分解成这三种基本结构的程序就不可能是结构化程序。结构化的程序只能有一个入口、一个出口，程序中没有永远执行不到的语句，不会形成死循环。结构化程序设计，实施方法体现在自顶向下、分块进行、逐步求精的程序设计方法。一方面，程序由三种基本结构组成；另一方面程序设计是采用逐层分解、一步一步精细化，直到每一步算法都具有有穷性、确定性为止。

现在，面向对象程序设计 OOP(Object Oriented Programming)和面向对象语言 OOL(Object Oriented Language)已经兴起，它提出了一种全新的程序设计思想，把数据和对数据进行的操作(代码)融为一体，从而可以完整地描述现实世界某一对象的全貌。面向对象程序设计方法包括：定义类程，建立属于类程的对象，以及基于已有类程派生新类程等。有人说：“80 年代面向对象技术将和 70 年代结构化程序设计一样得到发展。”面向对象技术是过程式设计技术的进一步发展，它是人们在解决软件危机中继结构化程序设计、软件工程之后的第三个关键步骤。面向对象的程序设计方法基本原理是对问题域的自然化分，按照人们自然思维方法对问题进行求解，并以对象为基础描述问题域。

C＋＋和 Turbo C＋＋是美国 Borland 公司推出的最新面向对象的程序设计语言，它继承并发挥了原来 Turbo C 集成环境的优良特性，包括面向对象程序设计的基本思想和设计方法，是目前最受欢迎的面向对象程序设计软件工具。用 Turbo C＋＋设计程序有利于计算

机系统软件和应用软件的开发，使软件生产变得相对容易，程序质量更加可靠。现在，随着面向对象程序设计的理论和方法的发展，面向对象的程序设计语言，如C++、Turbo C++也逐步流行起来，并大有迅猛发展的趋势。

9.2 开发程序系统的全过程

一个大型程序系统的研制与开发是一个十分复杂的过程，它可以分为以下五个阶段：程序系统的分析阶段；程序系统的设计阶段；程序系统的编码阶段；程序系统的调试、测试阶段；程序系统的验收、运行和维护阶段。每个阶段都有它的明确任务。从问题的产生、分析开始，设计编码、调试验收、运行维护、功能扩充直到最后系统失去使用价值，停止运行而报废，这就是整个程序系统的生命周期。下面我们对程序系统的每一个过程作些具体的分析。

(1)程序系统的分析阶段

首先要深入细致地调查问题、研究问题，搞清问题的来龙去脉，进行详细的可行性分析。其中包括：

①问题以什么形式提出，要解决些什么问题，如何解决这些问题。

②使用的计算机系统硬件、软件资源情况，如内、外存的容量，运算速度，外设的配置，是否要增加设备等。

③开发程序系统的代价分析，包括系统开发所花的时间、经费；系统实现后的社会效益、经济效益等。

④人员和组织分析，包括以前开发程序系统的经验教训，有关人员的水平和素质；现有的组织和管理能否达到科学化、规范化。详细并客观地进行分析，弄清楚有利因素和不利因素，做到心中有数。

⑤分析今后客观情况的变化。由于情况不断变化，目标任务可能有所变动，功能要求不断扩充。时间的推延，费用的增加，这些都是系统开始分析时应该考虑的问题。

在完成了这些初始阶段分析后，需要写出一份完整而准确的功能规格说明书或称为系统的可行性报告。

(2)程序系统的设计阶段

在功能规格说明的基础上进行总体设计、模块设计，建立程序的系统结构，包括数据结构和模块结构。传统的设计方法是先画粗框图，再画详细的程序流程图，最后用语言工具来编写程序。流程图比用文字描写一种算法、一个过程更加清晰、形象，使人一目了然。但是，现代程序设计方法对传统的流程图持不同意见，因为原来的流程图只重视了程序的流程，而忽视了程序的结构，现在人们主张用结构化程序图代替传统的流程图。做好了设计阶段的准备工作，就可以着手写出程序系统设计说明书。

(3)程序系统的编码阶段

根据模块说明和数据结构方法，按照一定的算法编写程序，这里包括程序设计语言的选择，程序风格的确定。这是将一个具体的物理问题转化为一个能在计算机上实现设计方案的过程。在选择程序设计语言时，尽量采用高级语言来编写程序，少用汇编语言。因为汇编语言包括与特定机器有关的内容和许多限制，可能会破坏清晰易懂的程序结构。在系统中，汇

编代码往往是在使用十分频繁、与硬件关系密切而又难于用高级语言描述的那些程序中使用，如系统初启、中断处理以及设备驱动等部分。

(4)程序系统的调试与测试阶段

程序编写出来了，上机不一定就能顺利通过；上机虽然暂时通过了，程序中也不一定就没有错误。程序员靠调试与测试来发现错误、排除错误，调试与测试工作贯穿在程序系统开发的过程中。调试、测试任务繁重，它的工作量可能占研制工作的 50%以上。飞行控制、核反应等程序系统调试、测试费用可能是系统研制开发的 3～5 倍。

实际上，程序系统的调试与测试只能发现程序中的错误而不能保证系统没有错误。E. W. Dijkstra 有句名言：“测试只能证明程序中还有错误。”程序系统的调试、测试是程序设计中的重要方法，是提高程序的正确率、保证程序能可靠运行的重要手段。

(5)程序系统的验收、运行和维护阶段

要达到程序系统完全正确是件不容易的事，那么程序验收的标准是什么呢?这一点值得讨论。当程序系统故障已经减少到系统能够多次正常运转，运转过程中还没有发现、发生故障和错误，运算结果经过分析也认为是可信的、正确的，这时程序系统就可以达到验收的要求了。

程序系统通过了验收，下一步是运行和维护阶段。由于系统运行环境的变化、运行过程中发生新的问题、用户提出新的要求等原因，这个阶段要对程序系统进行修理(repair)，更改(revisions)和增加(enhancements)等工作，即在程序系统运行过程中改正错误，修改规范、设计和程序，对新的要求和目标增加新的程序功能。简单地说，这个过程就是对系统进行再分析、再设计、再维护，反复交替进行的过程。

在完成了开发程序系统的过程后，程序设计人员要写出一份较详细的程序系统使用说明书，供用户使用、维护参考，并存档。

9.3 程序系统中错误类型分析

程序员编写程序，一要事先对问题进行详细地分析，搞清目标任务和各种主观客观情况，做到有的放矢；二要有一丝不苟的认真、严谨态度，绝不可以心血来潮、挥笔而就，那将使程序系统错误百出。

一个设计好的程序系统可能有这样那样的错误。我们要分析错误的类型，找出错误的原因，并用具体的实例说明系统中出错的地方。程序中错误一般可以分为三类：语法错误、语义错误和逻辑错误。在同一个程序例子中可能同时出现两种或两种以上的错误类型，一种错误类型也可能会多次出现。下面对程序中的错误类型作一些具体分析。

(1)语法错误

语法错误是指程序中语句、定义、说明以及程序本身不符合语法规则。如使用了没有定义的标识符，使用了字符集中没有的字符和一些已经规定的变量，循环中变量不一致，括号不匹配等。下面以例子来说明。

例 9.2　产生 10 个随机数，并打印产生随机数的顺序数和随机数。

```
! Create randomize.
! Count from 1 to 10 and print.
RANDOMIZE
FOR i = 1 TO 10
    LET a = INT(RAD * 90+1
    PRINT i;a;
NEXT i
END
...............
OK. run
Illegal expression.
```

检查程序第 4 行，发现括号不配对，改为：

```
a=INT(RAD * 90)+1
```

重新执行上述程序得到：

```
1  1  2  1  3  1  4  1  5  1  6  1  7  1  8  1  9  1  10  1
```

由结果可见，产生随机数的顺序是 1，2，…，10，而 a 取得的值总是 1。显然，虽然给产生随机数的公式“埋下”不同的种子，但所得结果都是 0，这说明取随机数时出了故障。经过检查，取随机数公式 RAD 函数写错了，应该是 RND。在执行程序时，机器把 RAD 当做变量了，不把它当作随机数发生器来产生随机数。RAD 若是个变量，没有给变量赋值，RAD 的值总是 0，所以 a 的值总是 1。这个例子中前一个错误是语法错误，后一个是语义错误。

下面再看两个带语法错误的赋值语句：

```
LETv=(4/3) * π * r
LET y=x-(6-x * (5-x * (4-3 * x))
```

这里前一个语句不仅写法错误，而且使用了字符集以外的字符 π，机器不能识别，后一个语句是少了一个右括号，产生括号不匹配。

另外，在 SWAP 语句中，如果被交换的两个变量类型不一致，就会出现数据类型不匹配(TYPE MISMATCH)错误，在程序中把保留字拼错，保留字作变量名，标点用错，DATA 语句中的数据和 READ 语句中的变量不匹配，程序结构错误(如 FOR 和 NEXT 不对应等)。

一般说来，语法错误比较简单，可用编译程序检查，有错时就显示出相应的错误信息。这种错误又称为编译型错误。

(2)语义错误

语义错误是程序系统中语义本身有错，如除式的除数为零，表达式的值超出设定范围，死循环等。

赋值语句 LET y1=100/sqr(1-x * x)，当 $|x|>1$ 时，开平方函数失去意义，出现了非法函数调用错误(ILLEGAL FUNCTION CALL)；当 $|x|=1$ 时，出现了除数为零的错误(DIVISION BY ZERO)。

再看一个循环程序的例子。

例 9.3

```
! An example of "repetition".
LET sum=0
FOR i = 1 TO 100
    LET i = i-1
    LET sum=sum+exp(i)
    PRINT sum;
NEXT i
END
```

执行这个程序就会发现，sum 的值不断增加，并且每循环一次增加 1。经过分析不难发现，这是个死循环程序，i 的值永远不会增加到 100，循环一直执行下去；另外 $e^0=1$，所以程序每循环一次，sum 的值增加 1。

一个一元二次方程 $ax^2+bx+c=0$ 的两个根应该是：

$$x_1=\frac{-b+\sqrt{b^2-4ac}}{2a}$$

$$x_2=\frac{-b-\sqrt{b^2-4ac}}{2a}$$

如果在程序中把一个根的赋值语句写成：

```
LET x=-b+sqr(b*b-4*a*c)/2*a
```

这种情况产生了语义错误，虽然计算结果出来了，可是这个结果并不是方程的根。语义错误常常不易被人们发现。

(3)逻辑错误

逻辑错误是指那些并没有违反语法规则，但是违反了逻辑规则，或出现了与题意根本不符的算法设计错误和算法实现错误。逻辑错误的出现大概有两种情况：一种可能是问题的分析、算法的错误；另一种可能是程序没有反映算法要求。如错写或遗漏了运算符，参数的类型、顺序、个数不一致，该执行的语句没有执行到等。例如，当 $y=f_1(x)-f_2(x)$ 的值很小时，继续用这个式子计算下去会产生精度丢失。解决这个问题的办法是用 Taylor 展开式：

$$y=f_1(x)-f_2(x)\approx h(f'_1(x_0)-f'_2(x_0))$$

其中，x_0 接近 x，h 的值充分小。

对计算 $y=\sqrt{a+1}-\sqrt{a}$ 的式子，可以改变算法：

$$y=\frac{1}{\sqrt{a+1}+\sqrt{a}}$$

例 9.4　计算二项式展开式的系数

$$C_m^n=\frac{m!}{n!(m-n)!}$$

```
! Calculate binomial coefficient.
SUB cmn(f,k)
    LET f=1
    FOR i=1 to k
        LETf=f*i
    NEXT i
END SUB
! Main program.
INPUT PROMPT"m,n=";m,n
LET l=m-n
CALL cmn(a,m)
CALL cmn(n,b)
CALL cmn(c$,l)
LET c=a/(b*c)
PRINT "c(";m;",";n;")=";c
END
………………
OK. run
```

出现了错误信息：

```
Wrongtype.
```

由于调用子程序时实参个数和形参个数虽然一样多，但它的顺序、类型和形参不一一对应，因此计算程序出现了逻辑错误。把 cmn(c$,l)改为 cmn(c,l)后执行：

```
OK. run
Division by zero.
```

再检查，发现 cmn(b,n)错写成了 cmn(n,b)，改正后就对了。再执行：

```
OK. run
m,n=10,8
```

```
c(10,8)=45
OK.
```

9.4 程序的调试与测试方法

在程序设计课程中，一方面要学习程序设计语言，学习程序设计方法，从而对提出的某一问题，在掌握了物理过程或工作状态的基础上，得出数学模型表达式，再简化数学模型，建立近似公式并确定数值计算方法后熟练地编写程序、上机计算；另一方面，要学习计算机基本知识，应用软件工具和方法来对程序系统进行调试与测试，努力提高程序的正确率。

(1)用调试、测试方法排除程序系统中的错误

程序员用调试和测试方法，可以发现和查找程序系统中的错误。对于一些大型复杂的程序系统，调试和测试甚至比分析、设计程序还要化更多的时间和费用。这项工作是程序开发中至关重要的环节，它能发现错误、排除错误，说明程序中存在错误，但又不能证明程序系统中就没有错误了。程序中语法错误一般在程序编译时就能发现，语义错误往往在程序运行时发现，逻辑错误又往往在系统运行后才能发现。值得注意的是，语义错误和逻辑错误一般较难发现，发现又难于定位和排错，这使得程序虽然编译通过了，而运行后不一定能得到预期的结果。对语义错误和逻辑错误，可以用系统提供的有关调试程序进行查找，借助于它追溯程序运行的路线，跟踪并输出一些变量的情况，进行仔细分析。举一些典型的例子，用调试、测试办法对系统进行分析是必要的，如产生死循环要检查循环的设置条件，结果出现上溢、下溢要检查乘除法，产生值越界要检查子域类型表达式的变化等。下面看一个例子。

求一个代数方程 $x^3-x-1=0$ 的根。可以用简单迭代法来解代数方程。首先把原方程变为下列等价形式：

$$x_{n+1}=x_n^3-1 \qquad (n=0,1,2,\cdots)$$

并取 $x_0=1.5$ 进行迭代，其结果：$x_1=2.375$，$x_2=12.39$，…，显然这样的迭代结果是发散的，迭代不出方程的根。迭代过程中可用跟踪方法看 x 值的变化情况。这里讨论下列几个问题：

①方程 $x^3-x-1=0$ 的等价方程如何求法。用 $x_{n+1}=x_n^3-1$ 为等价方程是不行的，改用 $x_{n+1}=\sqrt[3]{x_n+1}$ 作为等价方程，给 $x_0=1.5$，经过几次迭代得到：$x_1=1.35721$，…，$x_7=1.32472$，$x_8=1.32472$。显然，原方程的根是收敛的，它趋于 1.32472。

②精度要求和迭代次数。根据需要，用两次迭代之差小于某一个数 ε(充分小的数)来控制精度和迭代次数。另外，迭代次数还和 x_0 给得好坏有关。

③初根 x_0 的取法。我们可以采用逐步搜索法先找出有根的区间，然后尽可能找出离真根近的 x_0 为初始根。对于上述方程，取 $x_0=1.5$ 即可。

④几何意义。根据给出的等价方程，分析迭代求根的收敛和发散情况，从而来讨论迭代过程中的几何意义。

(2)测试方法简单分析

当用户只知道程序输入/输出结果，而不知道程序内部结构时，则可利用程序外部特性来测试程序，即黑盒子测试法。为了找出程序中不符合功能说明的全部错误，就非得进行穷举测试。当然，这种穷举法一般是很难做到的。另外，如果知道程序内部结构和路径，则将每

种可能的路径执行一次，这就是白盒子测试法。同样为了找出所有错误，只有进行穷举法执行所有路径。实际上，穷举路径测试次数可能是个天文数字，这是做不到的。所以说，这种方法不能证明程序的正确性。但是，经过测试了的程序总比没有测试过的程序相对正确一些，尽管这种测试不可能把所有执行路径都走到。

程序中逻辑错误不仅难于发现，而且往往容易被忽视，这种错误可以采用测试方法来检测。测试的办法之一是设计三类数据集合，即正确数据集合、错误数据集合和边界数据集合，分别用这些数据来检验程序运行是否达到预期的结果。例如编写一个求三角形面积的程序，我们可以送以下三类数据集合进行计算：

①正确的三条边：3,4,5　　4,5,3　　5,4,3

②错误的三条边：-1,-2,5　　4,-3,5　　3,4,10

③边界的三条边：1,2,3　　3,4,6.99　　3.1,4.2,7.31

上述三类数据集合送到机器里运行的结果是我们可以预期到的，从而可以来测试计算三角形面积的程序是否正确。如果测得程序有错，便可用调试的方法进行信息跟踪，具体情况具体分析，按照不同的跟踪信息，确定具体错误位置和错误性质，进行排错处理。

(3)**BREAK** 命令和 **CONTINUE** 命令

可以用程序调试命令 **BREAK** 和 **CONTINUE** 来调试程序。请看下例。

例 9.5　一个错误的例子。

```
! This is a erroneous program.
OPEN #1:printer
DIM x(6)
FOR i = 1 TO 10
    LET x(i) = i*i
    PRINT #1: x(i);",";i;";";
NEXT i
END
............
OK. run
Subscript out of bounds.
```

可以将断点设在第 4 行，运行程序：

```
OK. BREAK 4
```

断点已指示在第 4 行。

```
OK. run
Breakpoint.
```

再执行 CONT(CONTINUE 的缩写)命令：

```
OK. CONT
1,1;4,2;9,3;16,4;25,5;36,6;
Breakpoint.
Subscript out of bounds.
OK.
```

由上述运行结果，经分析得知：继续语句使程序继续执行到 i=6 后超越数组下标的上界 6(数组定义了 DIMx(6))而停止。

(4)**TRACE. TRU** 跟踪命令

使用跟踪调试工具 **TRACE. TRU** 命令，可以跟踪显示程序的动态执行路径，有利于发现程序中的逻辑错误。跟踪命令以 **DO** 命令方式执行，它的形式为：

DO TRACE,option(variables)

其中，option 选择项可以是 step(单步)、slow(慢步)、fast(快步)以及 break(断点跟踪)方式之一；variables 为跟踪变量名表，它不多于 8 个，变量多时用逗号“，”分开。用户在执行跟踪时，可以根据需要键入单键改变跟踪状态。如键入“S”键将跟踪状态改为“slow”，键入“F”改为“fast”，键入“空键”改为“step”，键入“Q”时停止跟踪。使用 BREAK 方式，开始为连续快速跟踪，直到条件为“真”时改为“step”跟踪。这种跟踪的形式为：

```
DO TRACE,BREAK(condition,variables)
```

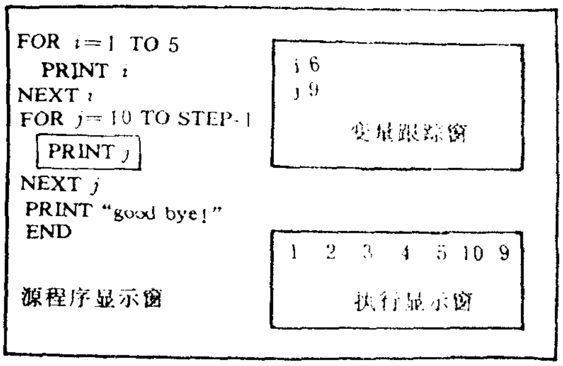
其中 condition 为跟踪条件。

下面举例说明跟踪方法。

例 9.6　编写分别打印 1～5 和 10～1 的各个整数。

```
! Print numbers 1 through 5.
FOR i = 1 TO 5
    PRINT i;
NEXT i
! Print numbers from 10 down to 1.
FOR j = 10 TO 1 STEP -1
    PRINT j;
NEXT j
PRINT "Goodbye!"
END
..............
OK. DO TRACE,SLOW(i,j)
```

执行跟踪命令时，屏幕显示分为三个窗口(如图　　左半边为跟踪源程序窗口，当前执行的高辉亮行追踪显示；右上半部为变量跟踪显示窗口；右下半部为执行显示窗口。



```
FOR i=1 TO 5
  PRINT i
NEXT i
FOR j=10 TO STEP-1
  PRINT j
NEXT j
PRINT "good bye!"
END
```

```
i 6
j 9
```

```
1  2  3  4  5 10  9
```

图 9.1　**跟踪显示屏**

一个大型程序，可以首先一个模块一个模块地分别调试，在调试时要作一些必要的处理，如在需要的地方适当加上输入/输出语句。一旦模块程序调试完毕，应该恢复程序原样。当各个分块程序调试正确后，可以对整个程序进行统调。统调时除注意各个模块的连接外，还应注意参数的传送、模块变量不能互相干扰。总之，单个模块调试和程序统调都要反复进行，反复验证，直到最后程序运行结果可信、正确为止。

调试和测试是验证程序正确与否的重要方法。现在，程序系统正确性证明尚无有效且成熟可行的方法，一个经验丰富的程序员只有把程序调试、测试的各种方法结合起来，综合应用，具体问题认真分析，才能确保程序系统的正确性。

9.5　结构化程序设计方法

结构化程序设计(Structured Programming)是一种先进的程序设计方法，它有顺序(Sequence)、选择(Selection)和循环(Repetition)三种基本控制结构(如图 9.2 所示)，并且程序只有一个入口和一个出口。按照结构化程序设计方法编写出来的程序层次分明、简洁、易读、易维护，程序便于调试与验证。结构化程序设计方法可以包括以下四个方面的内容。

9.5.1　限制使用 GOTO 语句

E. W. Dijkstra 首先提出了结构化程序设计的概念，并在 1965 年召开的 IFIP(国际信息

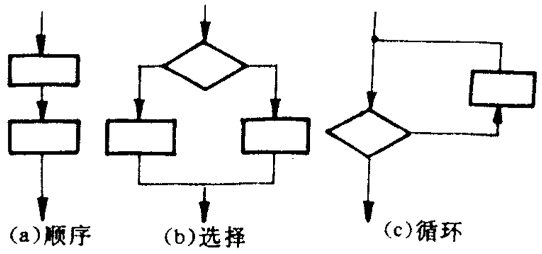

(a)顺序 (b)选择 (c)循环

图 9.2 结构化程序三种基本结构

处理联合会)会议上第一次提出了“GOTO 语句可以从高级语言中取消”,“一个程序的质量与程序中所含的 GOTO 语句的数量成反比”的观点。E. W. Dijkstra 还在 CAM 杂志上发表了名为《GOTO 语句看来是有害的》短文,他认为 GOTO 语句太容易把程序搞乱,应从一切高级语言中去掉;只要有顺序、选择、循环三种基本构件或通过它们的复合,就可以写好各种程序。1972 年 IBM 的 Mills 又提出结构化程序应该只有一个入口和一个出口,进一步阐明了结构化程序设计的规则。有人说,E. W. Dijkstra 的理论对传统的程序设计方法来说是“捅了马蜂窝,引起了轩然大波”。因为当时的高级语言和一些正在运行的程序系统都使用了 GOTO 语句,要取消 GOTO 语句将会使很多程序无法运行。

1974 年,D. E. Knuth 发表了题为《带 GOTO 语句的结构程序设计》的文章。他指出若在程序中不加限制地使用 GOTO 语句,程序系统无规则地跳来跳去,会使程序难以阅读和理解,易出错,更难调试和查错,所以应该限制使用 GOTO 语句。在不破坏程序良好结构前提下,有控制地使用 GOTO 语句,可以提高程序效率。结构化程序设计方法并不完全禁止使用 GOTO 语句,只是反对滥用 GOTO 语句。应该限制 GOTO 语句使用,只有控制转移仅限于本块结构范围内时才不得不使用 GOTO 语句。这时使用的 GOTO 语句既不破坏程序结构,又使程序显得简洁,使程序的静态结构和动态结构相差不大;程序还是一个入口和一个出口的结构。实践证明,少用 GOTO 语句,是结构化程序设计重要内容之一。

用 True BASIC 设计程序可以不使用 GOTO 语句,不使用行号。如果一定要用到 GOTO 语句,则必须在有关语句前加上行号。True BASIC 提供了一种规范化的 GOTO 语句——EXIT 语句。EXIT 只能从一个控制结构内部转到这个控制结构的出口处,而不能像 GOTO 语句那样上下随意跳动。值得注意的是,只有在特殊情况下才使用 EXITIF、EXITDO 以及 EXITFOR 语句。它们可能破坏了一个出口的原则,但使用了可以提高程序效率,增加可读性,这里只能根据 True BASIC 的特点,扬长避短,形成具有 True BASIC 的特定程序风格。

9.5.2 结构化程序设计的三种基本结构

结构化程序设计的三种基本结构是顺序结构、选择结构和循环结构。这就是说结构化程序可以由上述三种基本结构组成,较复杂的结构化程序由这三种基本结构复合、叠加而成。

(1)顺序结构。顺序结构是结构程序最简单、最基本的结构,如图 9.3 所示。序结构。①是入口,②是出口,各块是按出现的顺序执行的。a 块后面是 b 块,b 块内又有 b_1、b_2 块。大块内可能有小块,小块内不会有控制转移语句。总之,一个顺序结构是由许多顺序执行的语句组成,这些可执行语句是依次顺序执行下去的。

(2)选择结构。选择结构又称为分支选择结构,如图 9.4 所示,它仍然只有一个入口和一个出口。它按照给定的条件来判断在两条不同的路径中选择哪一条,然后再分别作 a 块或 b 块处理。程序中的“块”是一个不产生控制转移的执行语句,或者是无任何执行语句的“空块”,或者是结构化程序三种基本结构的组合。一个执行语句和一个空块称为“基本块”,而顺序、选择、循环三种基本结构以及它们的组合称为“复合块”。复合块是按照一定的顺序和逻辑关系组合起来的。有了空块的概念,我们可以认为选择结构图中 a 块和 b 块之一可

能不做任何工作，称之为“空块”。综上所述，一个块（a 块或 b 块）只包含一个执行语句是最简单的块；一个块也可以由若干条语句组成。

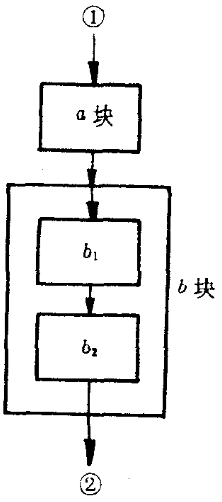

①
a 块
b_1
b_2
b 块
②

图 9.3　**顺序结构**

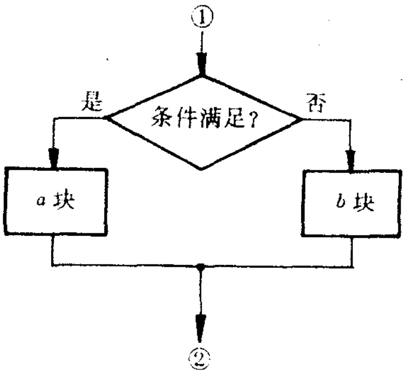

①
是
条件满足？
否
a 块
b 块
②

图 9.4　**选择结构**

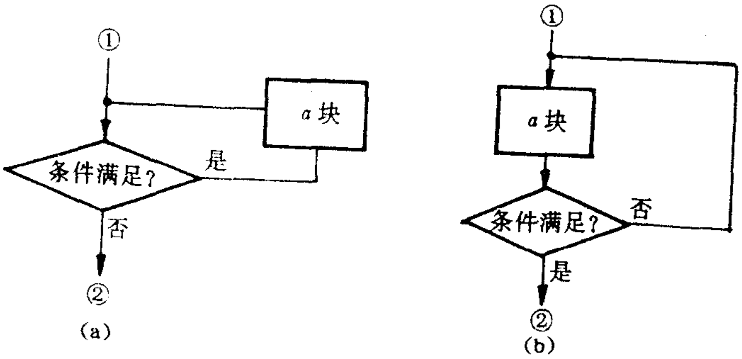

①
a 块
条件满足？
是
否
②
(a)
①
a 块
条件满足？
否
是
②
(b)

图 9.5　**循环结构**

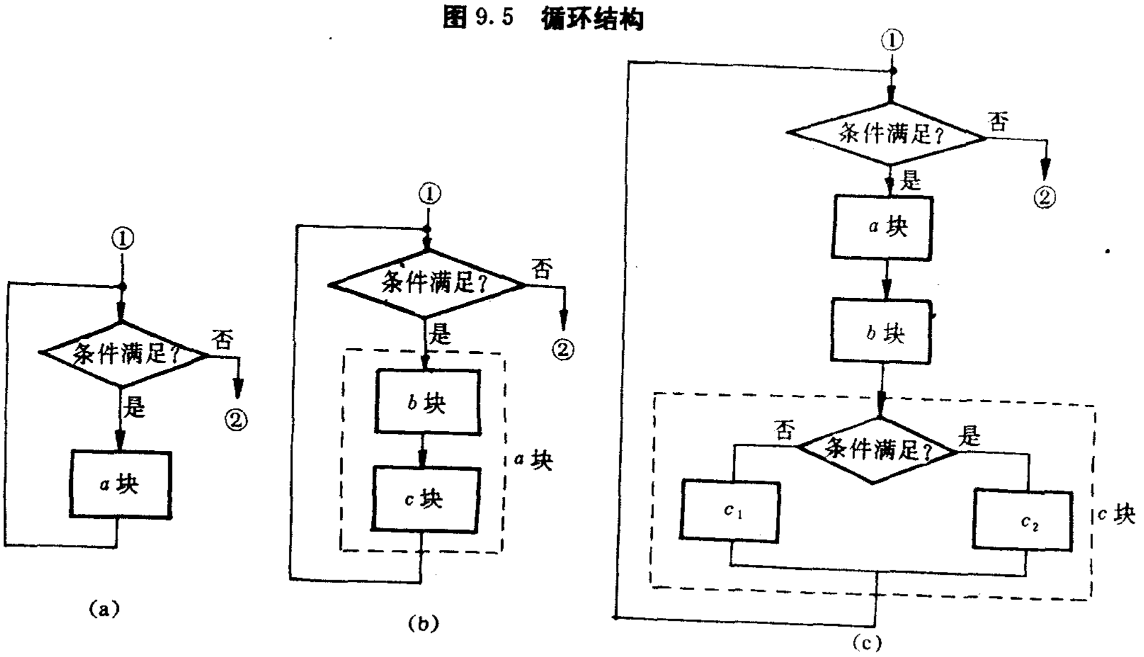

①
条件满足？
否
是
②
a 块
(a)
①
条件满足？
否
是
②
b 块
c 块
a 块
(b)
①
条件满足？
否
是
②
a 块
b 块
否
条件满足？
是
c_1
c_2
c 块
(c)

图 9.6　**顺序、选择、循环结构分析图**

（3）循环结构。循环结构是根据条件来判断是否要重复执行某些块的结构。一种情况是

当条件满足时反复执行 *a* 块，否则不执行 *a* 块，如图 9.5(a)所示。另一种情况是先执行 *a* 块程序，然后按条件判断，条件不满足时反复执行 *a* 块，而条件满足时则不执行 *a* 块，如图 9.5(b)所示。这两种不同情况的循环结构也只有一个入口和一个出口。另外注意这两种循环结构不同之处：①前一种情况是先按条件判断，满足条件时执行 *a* 块，即“先判断，后执行”，*a* 块可能一次也不被执行。后一种是“先执行，后判断”，*a* 块至少被执行一次。②前一种情况是条件不满足时退出循环，后一种是条件满足时退出循环。现在已经证明，一个结构化的程序只有一个入口和一个出口；每个语句都应当有一条从入口到出口的路径通过它，没有永远执行不到的语句；没有死循环程序。满足这些条件的程序一定是由上述三种基本结构所组成。反之，任何一个结构化程序都可以被分解为上述三种基本结构。图 9.6(a)是一个循环结构。图中 *a* 块又包含 *b* 块和 *c* 块[图 9.6(b)]，这里 *b* 块和 *c* 块又是一个顺序结构。图 9.6(c)中，c 块又包含了一个分支选择结构。显然，这是一个结构化程序的流程图，它是由顺序结构、选择结构和循环结构组成的，只有一个入口和一个出口。

9.5.3 结构化流程图

上面用传统的流程图表示了结构化程序的三种基本结构。传统流程图的缺点一是不能保证程序的结构化，二是当程序系统复杂时流程图的篇幅较大，作图也要花费较多的时间。结构化流程图象一个多层的盒子，所以又称盒图(box diagram)。盒图是美国学者 I. Nassi 和 B. Shneiderman 于 1973 年提出来的新的流程图工具，有人用他们两人名字开头字母命名，所以又称 N—S 图。结构流程图不用箭头来表示程序的走向，因此看流程图时不需要沿着流程图的方向上下左右来回跟踪寻找，看流程图如同看一本书，在一个大的方框里一看就明白程序的执行路线了。这里根据结构化程序的三种基本结构介绍结构化流程图。

(1)顺序结构图。图 9.7 是顺序结构图。按顺序，先执行 *a* 块，再执行 *b* 块。

(2)选择结构图。图 9.8 是选择结构图。如果条件为真，则执行 *a* 块，否则执行 *b* 块。其中 *a* 块、*b* 块之一可能是无任何执行语句的空块。

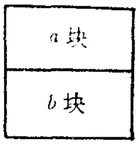


图 9.7 **顺序结构**

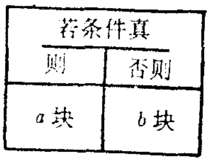


图 9.8 **选择结构**

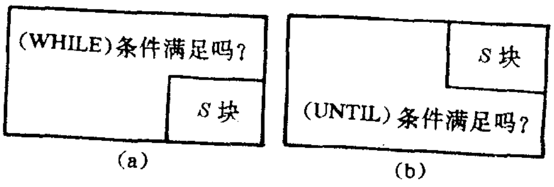


图 9.9 **循环结构**

(3)循环结构图。图 9.9(a)和(b)分别是包含 L 形框的循环结构图。循环条件在 L 形框内，需要重复执行的内容是小矩形块，即 *S* 块。前者是“当型”(WHILE)循环，它是“先判断，后执行”，而后者是“直到型”(UNTIL)循环，它是“先执行，后判断”。图 9.9(a)和(b)分别对应图 9.5(a)和(b)的一般流程图。

9.5.4 自顶向下逐步求精

以前，程序员通常先编写程序，再进行调试，在程序调试的过程中找出程序的故障并加以修改。这个过程说明了程序中有错误而不能证明程序中再没有错误了。E. W. Dijkstra 关于程序正确性证明曾论述过：“不应该先编出程序，然后再证明正确性。因为这样做只会增加可怜的程序员的负担。相反，程序员应该使正确性证明和程序编制彼此相关地同时进行。”

这里再强调一下，程序中过多地使用了 GOTO 语句，使程序系统的静态结构和动态执行情况相差很大。程序系统执行路线很难跟踪、检索，这种程序质量低劣，层次模糊，语句紊

乱，没有条理，难读、难理解，容易出错，更难调试与查错。

采用结构化程序设计方法，可以提高程序设计质量，解决程序层次不清问题。结构化程序设计方法实际上包括了算法设计、数据结构设计和程序结构设计。算法设计、数据结构设计的方法可以参考数值方法和数据结构有关资料。结构设计我们在前面已经介绍了三种基本结构，这里讨论的是结构化设计实施的具体原则一自顶向下(Top to down)逐步求精(Step－nice refinement)的方法。

程序设计的过程应该是：对一个具体问题，从最顶层开始分析研究，一层一层向下展开，逐步分细，一步一步精细化，直到它们能用具体的语句描述出来为止。要解决一个问题，首先把这个系统的来龙去脉进行详细研究，从总体到局部进行分析、分解，进行理论化、抽象化。从总体的抽象算法到具体的抽象操作，完成一个从“做什么”到“怎样做”的完整分析过程。经过对一个复杂系统的分析、分解和抽象，可以“自顶向下，分而治之”，进行“模块设计，结构化编码”，使程序系统的设计工作变得简单方便。著名计算机专家 N. wirth 教授曾说过：“程序就是在数据某些特定的表示方式和结构的基础上对抽象算法的具体表述。”

自顶向下逐步求精的设计方法，首先考虑了最能反映问题本质的总体要求，从最高级抽象概念出发，再进行分级抽象，把系统逐步细化求精，分解成一层层一个个功能模块，最后用某种程序设计语言编写出可以在机器上运行的具体程序。

一个大的程序系统可以分解为若干模块。设计从总体模块开始，自顶向下逐步求精，它是以模块化为基础，分阶段展开程序逻辑的设计方法，因此程序系统可以得到良好的结构。用这种方法编写程序，程序员一边设计程序，一边验证程序的正确性，可以把程序故障消灭在程序设计过程中，而程序的调试和测试仅仅是提高程序正确性和可靠性的辅助手段。

9.5.5 结构化程序设计方法举例

这里用两个具体的例子说明结构化程序设计方法。

例 9.7　读入一批正整数，并把其中最小的数打印出来。

顶层设计：首先从顶层开始把问题分解成两部分：(1)找出最小数；(2)打印最小数。

第二层设计：能细分的部分再细分：

(1)找出最小的数。①用 min 代表最小的数，并先充 0；②逐步输入需要比较的正整数，把最小的数存入 min 中。

(2)打印 min 的内容。

第三层设计：分析第二层设计中(1)的第②部分，还可以精细化为：

(1) 找出最小的数。①将 min 充 0；②对输入的数进行比较，最小的数存入 min 中，(a)输入一个数；(b)如果这个数小于 min 中的数，则将这个数送 min，否则做下一步。

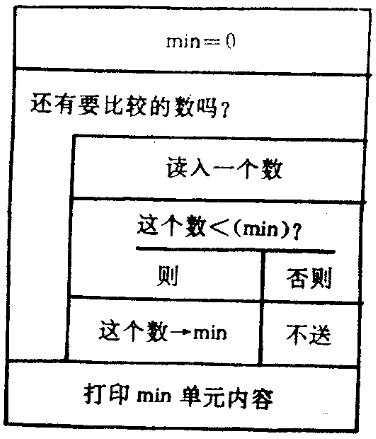


图 9.10　结构化程序设计图

(2)打印 min 内容。

根据上述分析，画出结构流程图。如图 9.10 所示。

这个例题的 Ture BASIC 程序比较简单，请读者自己写出。

例 9.8 给定一组系数 a,b,c,求一元二次方程:$ax+bx+c=0$ 的根。下面用结构化程序设计方法进行分析。

顶层设计。根据题意,求 $ax^2+bx+c=0$ 的解,总的分为下列三种情况:

(1)读入一元二次方程 $ax^2+bx+c=0$ 的系数 a,b,c 的值;

(2)印出要解的方程;

(3)方程 $ax^2+bx+c=0$ 的根为:

$$x_1=(-b+\sqrt{b^2-4ac})/(2a)$$

$$x_2=(-b-\sqrt{b^2-4ac})/(2a)$$

第二层设计。顶层设计的 1 和 2 不变,将(3)细分:

①当 $a=0$,原方程变为线性方程 $bx+c=0$,根为 $x=-c/b$;

②当 $a>0$,方程的根有 x_1 和 x_2。

第三层设计。前面的(1)和(2)不变,还是精细化(3)的内容:

①当 $a=0$ 时,又分为两种情况:

(a)当 $b=0$ 时,方程无根;

(b)当 $b>0$ 时,方程有一根,即 $x=-c/b$。

②当 $a<>0$ 时,又分为:

(a)当 $b^2-4ac\geqslant0$ 时,有两个实根 x_1 和 x_2(如果 $b^2-4ac=0$,则 $x_1=x_2$)

(b)当 $b_2-4ac<0$ 时,有两个复根:

$$x_1=-\frac{b}{2a}+(\sqrt{-b^2+4ac})i$$

$$x_2=-\frac{b}{2a}-(\sqrt{-b^2+4ac})i$$

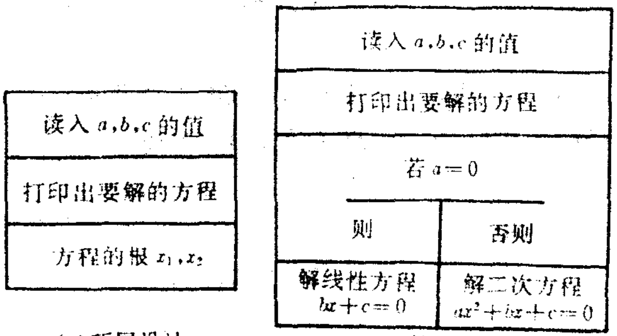
在这一层设计中,当 $b=0$ 时,还可以考虑 $c=0$ 和 $c<>0$ 的情况。另外,当 $b_2-4ac>0$ 时有两个根,$b^2-4ac=0$ 时只有一个根,这从根的求解公式中就清楚了,所以不必再精细化分了。

根据上述自顶向下逐步求精的设计方法,可以写出 True BASIC 程序如下:

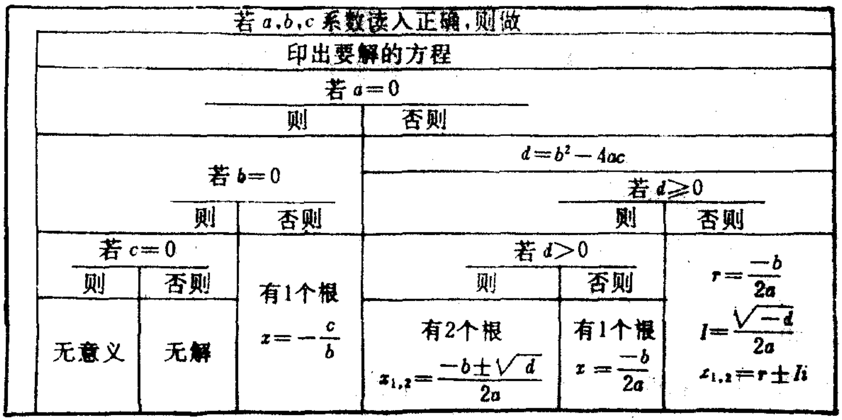
```
! A example of True BASIC program.
INPUT Prompt "输入系数 a,b,c:":a,b,c
IF a=0 THEN
  IF b=0 THEN
     PRINT"方程无意义!"
  ELSE
       LETx=-c/b
       PRINT"方程有一根:";x
  END IF
  ELSE
  LET d=b*b-4*a*c
  IF d>=0THEN
        LET x1=(-b+sqr(d))/(2*a)
        LET x2=(-b-sqr(d))/(2*a)
        PRINT"方程有实根:";x1;x2
      ELSE
        LET r=-b/(2*a)
        LET i=sqr(-d)/(2*a)
        PRINT"方程有两个复根:"
        PRINT r;"+";i;"i",r;"-";i;"i"
      END IF
    END IF
    END
```

再看运行结果:

```
OK. run
```


读入 a,b,c 的值
打印出要解的方程
方程的根 x_1,x_2
读入 a,b,c 的值
打印出要解的方程
若 $a=0$
则
否则
解线性方程 $bx+c=0$
解二次方程 $ax^2+bx+c=0$

(a)顶层设计　　(b)第二层设计


若 a,b,c 系数读入正确，则做
印出要解的方程
若 $a=0$
则
否则
若 $b=0$
则
否则
$d=b^2-4ac$
若 $d\geqslant0$
则
否则
若 $c=0$
则
否则
无意义
无解
有1个根 $x=-\frac{c}{b}$
若 $d>0$
则
否则
有2个根 $x_{1,2}=\frac{-b\pm\sqrt{d}}{2a}$
有1个根 $x=\frac{-b}{2a}$
$r=\frac{-b}{2a}$
$I=\frac{\sqrt{-d}}{2a}$
$x_{1,2}=r\pm Ii$

(c)第三层设计

图 9.11　求 $ax^2+bx+c=0$ 根的自顶向下设计方法

```
输入系数 a,b,c:0,0,9,方程无意义!
OK. run
输入系数 a,b,c:0,2,3,方程有一根:-1.5。
OK. run
输入系数 a,b,c:1,18,81,方程有实根:-9.0000-9.0000。
OK. run
输入系数 a,b,c,:1,4,1,方程有实根:-.267949-3.73205。
OK. run
输入系数 a,b,c:3,-5,7,方程有两个复根:
.833333 + 1.28019 i .833333 - 1.28019 i
OK. run
输入系数 a,b,c:1,9。
Too few input items. Please add more.
? -18
方程有实根:1.48466-10.6847,补上一个数就可以了。
```

小　结

本章首先介绍了什么是程序，什么是程序设计，重点讨论了结构化程序设计问题，目的

是要用先进的程序设计理论和方法去指导程序设计工作。通过本章学习，应该掌握以下几个问题：

1. 一个程序系统的好坏是有一定的标准的。

2. 从程序系统的分析研制、调试运行，直至程序系统的“死亡”是开发程序系统的全过程。这就是一个大型复杂的程序系统的生命周期。

3. 了解程序系统中可能存在的三种错误类型，并掌握一些程序调试与测试的方法和工具。

4. 学习结构化程序设计方法：程序中应限制 GOTO 语句的使用；程序的三种基本结构；结构化流程图；自顶向下逐步求精的方法。

习　题

1. 什么是程序？什么是程序设计？

2. 怎样理解 60 年代的“软件危机”？当前是否存在“软件危机”问题？

3. 简述结构化程序设计方法。结构化程序设计有些什么特点？

4. 叙述程序系统的生命周期。

5. 程序中可能存在几种类型的错误？如何分析它们？

6. 如何对程序系统进行调试与测试？举例说明程序调试与测试的方法和工具。

7. “GOTO 语句是程序错误的祸根”这句话对吗？如何理解“一个程序的质量与程序中所含的 GOTO 语句的数量成反比”这句名言？

8. 试述结构化程序设计的三种基本结构，并画出它们的传统流程图和结构化流程图，从而说明结构化流程图的优点。

9. WHILE 型循环和 UNTIL 型循环有什么区别？

10. 简述自顶向下逐步求精的程序设计基本思想，使用它应遵循哪些原则？

11. 试用自顶向下逐步求精的程序设计方法和模块化菜单技术，设计学生成绩管理系统。功能包括：

(1)计算各人平均成绩，并打印各人成绩单；

(2)计算全班各科平均成绩和总平均成绩，并打印总表；

(3)成绩分档统计并输出。

12. 你已经学习过结构化程序设计语言——True BASIC，能否从结构化程序设计的角度来评价它。

第10章　使用与实验指导

10.1　MS－DOS操作系统使用实验

DOS(DISK OPERATING SYSTEM)——磁盘操作系统，是IBM－PC机的用户界面，是计算机的最基础的软件，主要功能是文件管理和输入输出设备管理，不负责具体的数据处理工作。用户用键盘命令或程序经操作系统控制、管理计算机的硬件、软件，使用计算机的信息资源。

(1)MS－DOS启动和退出

系统软盘插入A驱动器，关闭驱动器门，开机。开机程序为：先外部设备，后主机。稍等，显示器屏幕出现系统日期、时间，键入当前日期、时间，或按两次回车键，显示器屏幕出现：

```
A>—
```

>是DOS标志符，表示系统已经装入内存储器，等待用户键入系统命令。A是系统当前读、写驱动器。当前读、写驱动器可用盘符命令改变

```
A>B:
B>—
```

改变B为当前读、写驱动器。

MS－DOS运行结束，关机退出。关机程序为先主机，后外部设备。

系统复位(热启动)：CTRL＋ALT＋DEL

(2)MS－DOS命令

DOS命令分成内部、外部和批命令三类。内部命令为驻留在内存的命令；外部命令为驻留在磁盘的文件，运行外部命令时，先将命令程序装入内存，运行结束归还内存空间；批命令为一批DOS内部、外部命令组成的文件，扩展名为.BAT，执行批文件时，文件中的命令自动按顺序执行。

10.1.1　MS－DOS文件操作命令实验

(1)预习知识：文件和文件引用名

文件是存储在外存储设备(磁盘)上的信息集合。按存储的信息分为程序文件和数据文件两类。文件由操作系统管理，每个文件都有一个标识符，称文件引用名。文件引用名由四部分组成：

[<盘符:>][<目录路径>]<文件名>[<.扩展名>]

其中，盘符A或B指软盘驱动器，C或D指硬盘驱动器，如果文件就在当前驱动器的盘上，则盘符可以省略；文件在当前目录中，目录路径可以省略；文件名最长允许有8个字符，至少1个字符。扩展名用作指定文件的类别，最多3个字符。

文件目录项：文件在目录中登录的有：文件名、扩展名、文件长度、文件建立日期、时间五项。

(2)实验目的：掌握 MS－DOS 的文件操作命令 COPY、REM、DEL；文件目录显示命令 DIR；DISKCOPY、DISKCOMP 命令；软盘格式化方法。

(3)实验条件：两个软盘驱动器主机；DOS 系统软盘；实验用软盘一张

(4)实验步骤

①开机，启动 MS－DOS。DOS 系统软盘插入 A 驱动器，关闭驱动器门。开机，稍等，显示器屏幕出现系统日期、时间，键入当前日期、时间，或按两次回车键，显示器屏幕出现：

```
A>—
```

②文件目录显示

```
A>DIR             显示 A 盘文件目录项
A>DIR/P           分页显示 A 盘文件目录项
A>DIR/W           宽行显示 A 盘文件名、扩展名
A>DIR  F*.*       显示文件名 F 开头的文件目录项
```

③软盘格式化。实验用软盘插入 B 驱动器，关闭驱动器门。

```
A>DIR  B:          显示 B 盘文件目录项
A>FORMATB:         A 盘 FORMAT 外部命令装入内存，格式化
                   B 驱动器中的实验用软盘
A>DIRB:            显示 B 盘文件目录项，空盘
```

④整盘文件拷贝

```
A>DISKCOPY A:A:          单驱动器按提示整盘拷贝
A>DISKCOMP A:A:          单驱动器整盘比较
A>DIR                    显示练习盘文件目录
```

DOS 系统软盘插入 A 驱动器、实验用软盘插入 B 驱动器，关闭驱动器门。

```
A>FORMAT B:/S            带系统又一次格式化实验用软盘
A>DIR B:                 显示 B 盘目录，有一个 COMMAND.COM 文件
```

⑤文件拷贝

```
A>COPY FORMAT.COM B:               A 盘 FORMAT.COM 文件拷贝到 B 盘
A>DIR B:                           显示 B 盘目录，检查、证实之
A>COPY  M*.* B:                    A 盘 M*.* 文件拷贝到 B 盘
A>COPY FORMAT.COM B:ABC.COM        A 盘 FORMAT.COM 文件拷贝到 B 盘，
                                   改称 ABC.COM 文件
```

⑥文件改名

```
A>REN B:ABC.COM FORMT1.COM        ABC.COM 文件改名 FORMT1.COM 文件
A>B:                                改变 B 为当前读、写驱动器
B>REN FORMAT.COM FORMT2.ABC      B 盘 FORMAT.COM 文件改名 FORMT2.COM
B>DIR                                 显示 B 盘目录，检查、证实之
```

⑦文件删除

```
B>DELCOMMAND.COM            删除 B 盘 COMMAND.COM 文件
B>DIR                       显示 B 盘目录，检查、证实之
B>DELF*.*                   删除 B 盘 F*.* 文件
B>DIR                       显示 B 盘目录，检查、证实之
```

(5)思考题

①什么是当前读、写驱动器？改变当前驱动器的作用是什么？

②软盘为什么要格式化？软盘格式化应注意什么？

③文件多义符＊、？代表什么？

10.1.2　MS—DOS 目录操作命令实验

(1)预习知识：目录树结构。MS—DOS 的文件系统采用树结构，根结点表示根目录，树枝结点表示子目录，树叶结点表示文件。一张盘只有一个根目录，它在磁盘初始化时自动建立。文件操作命令只对当前目录有效，如果文件不在当前目录中，须指定文件路径或改变当前目录。系统启动后，当前目录为根目录。

(2)实验目的：掌握 MS—DOS 的目录操作命令 CD、MD、RD。

(3)实验条件：两个软盘驱动器主机；DOS 系统软盘；实验用软盘一张。

(4)实验步骤

①开机，启动 MS—DOS。当前驱动器改变 B，实验用软盘插入 B 驱动器，关闭驱动器门。

②当前目录显示和建立、改变当前目录

B>CD	显示当前目录，为 B：　根目录
B>MD\TBASIC	在根目录下建立子目录 TBASIC
B>COPY　A:COMMAND. COM\TBASIC	A 盘 COMMAND. COM 文件拷贝到 B 盘的子目录 TBASIC 下
B>DIR	显示当前目录，在根目录下看不见子目录 TBASIC 下的文件目录
B>DIR\TBASIC	显示、证实子目录 TBASIC 下有 COMMAND. COM 的文件
B>CD TBASIC	改变 TBASIC 为当前目录
B>DIR	显示当前目录，看见有 COMMAN. COM 文件
B>MD WANG	在目录 TBASIC 建立子目录 WANG
B>CD WANG	改变 WANG 为当前目录
B>COPY\TBASIC\COMMAND. COM	将子目录 TBASIC 下的文件 COMMAND. COM 拷贝到子目录 WANG 下
B>DIR	显示、证实子目录 WANG 下有 COMMAND. COM
B>CD\	改变 B:\为当前目录

③删除子目录

B>CD\TBASIC\WANG	改变 WANG 为当前目录
B>CD	显示当前目录为 B:\TBASIC\WANG
B>DEL COMAND. COM	删除当前目录下的文件 COMMAND. COM
B>DIR	显示当前目录为空目录
B>CD..	改变 TBASIC 为当前目录
B>RD WANG	删除当前目录下的空目录 WANG
B>A:TREE	显示 B 盘目录结构

(5)思考题

①什么是目录树结构？什么是根目录、子目录、当前目录？

②什么是目录路径？什么是文件路径？什么是绝对路径、相对路径？改变当前目录有什么作用？

③子目录删除的条件是什么？

10.2 True BSAIC 程序设计

10.2.1 单语句立即方式

(1)预习知识

① True BASIC 启动和退出

True BASIC 在 MS－DOS 2.0 或更高版本的支持下工作。当机器在 DOS 状态下，将 True BASIC 语言系统软盘插入 A 驱动器，在 A>提示符后，键入“HELLO”命令，按回车键后，稍等。True BASIC 启动后，显示器屏幕自动清屏，并分成上、下两部分。上部称编辑窗口，下部称背景窗口。在背景窗口中显示：

```
TRUE BASIC HERE
VERSION 1.0
COFYRIGHT(C) 1985 BYTRUE BASIC,INC
PUBLISHED BY ADDISION－WESLEY PUBLISHING COMPANY,INC
OK.
```

背景窗最下部出现 OK 及光标，OK 是标志符，表示 True BASIC 语言系统已经装入内存储器，等待用户键入语言系统命令。所有的系统命令均在背景窗最下部 OK 标志符后输入，立即执行，所以这一行称为命令行。

编辑窗口用作建立、编辑 True BASIC 源程序或文本；背景窗口用作显示输入系统命令、程序运行结果等历史背景。在进入 True BASIC 时，编辑窗口占 17 行(包括窗口分隔线占一行)，背景窗口占 7 行。在西文状态下，允许用户根据需要，用 SPLIT 命令改变两个窗口的划分。如果在命令行键入：

```
OK. SPLIT 9
```

命令系统按编辑窗口占 9 行，背景窗口占 15 行重新划分。SPLIT 命令后的数值是指编辑窗口占用屏幕的行数，它在 0 到 24 之间取值，0 表示将整个屏幕全部归背景窗口；24 表示整屏幕都划归编辑窗，背景窗只留一行命令行；当数值大于 24 时，出现出错提示“TRY SPLIT UPTO SPLIT 24”，因为背景窗口中至少应有一行命令行，下达系统命令。

在 CCDOS 支持下，允许使用汉字。当 True BASIC 调入内存后，在命令行键入高分辨模式命令：

```
OK.MODE HIRES
```

屏幕处于中文显示状态(对长城 0520 只要将其置于高分下状态就可)。

两个窗口间的切换，使用 F1 和 F2 功能键，F1 光标进入编辑窗；F2 光标进入背景窗。

在程序运行结束，需要退出 True BASIC 时，在命令行键入 BYE 命令，返回 DOS 状态。

② 语句命令。系统命令包括有程序文件处理命令、程序或文本编辑命令、辅助命令、程序调试命令和语句命令。

绝大多数 True BASIC 的可执行单行语句都可以作系统命令用，但非执行语句(如 DIM，DEF，OPTION，DATA 等)以及选择、循环结构不能作命令用。

可作系统命令用的语句又分两类：一类是任何时都可作命令使用的语句：

ASK　BOX　CLEAR　GET　LET
INPUT　LINE INPUT　PAUSE　PLAY　PLOT
PRINT　SET　SOUND

另一类是仅当未编译的程序运行完可作命令使用的语句：

CALL　CLOSE　DRAW　ERASE　MAT
OPEN　READ　RESET　RESTORE　WRITE
WINDON

即在程序运行结束后，语句命令才能使用子程序、子图，数据表，数组和通道，否则会搅乱中断程序的继续运行，或影响结果的可靠性。

③ 单语句立即方式。一般 BASIC 语言程序采用解释执行方式，True BASIC 语言程序可采用解释执行方式，或编译执行方式。在解释执行方式下，又分为：单语句立即方式——单行语句作系统命令用，用户输入一条语句，立即解释，执行见效；程序方式——一批语句一起解释执行，获得程序结果。立即方式又称交互执行方式，一旦句法出错立即提示错误信息，相当在计算机辅导下学习语言，初学者学习、理解 BASIC 句法功能十分方便；立即方式有助于初学者对程序概念的理解；在进行程序设计时，它还可用作程序调试。

(2)实验目的

①掌握 True BASIC 语言的基本类型数据、数据操作及表达式

②掌握 LET，PRINT 语句句法、作用，用单命令立即方式完成简单算术运算。

(3)实验范例：计算半径为 10 厘米的圆面积及圆周长。

(4)实验步骤

①开机

②启动 True BASIC，在命令行输入立即执行语句，每个语输入结束，按回车键(“ENTER”)光标自动回车换行，语句立即执行。

```
OK. LET r=10
OK. LET c=2 * PI * r
OK. LET s=PI * r * r
OK. PRINT  "Area=";s;"(cm^ 2)","cycle=";c;"(cm)"
Area=  314.159  (cm^ 2)  cycle=  62.8319  (cm)
OK.
```

(5)上机例题

① 已知三角形三边边长 $a=3,b=4,c=5$ 厘米，求三角形面积 A。

②已知政治成绩 $P=85.5$，英语 $E=77$，数学 $M=90$，求总分及平均成绩。

(6)思考题

①为什么说计算机是数据处理机？为什么说计算机解题程序是数据输入—处理—结果输出？如果变量不先赋值，以上范例会有什么结果？为什么？

②变量和数学中未知数有什么区别？归纳变量的作用。

③归纳两类数据的说明，操作及表达式，数值处理函数，和运算优先序。说明书写正确表达式的重要性。

10.2.2 简单程序

(1) 预习知识

① 源程序、程序编辑和运行

程序是用 True BASIC 语句书写的解题过程的全文。True BASIC 源程序是 ASCII 编码语句行的集合，一行一个语句，由 END 语句结束。所谓简单程序，它解决的问题相对比较简单；程序只是由一些输入、输出或简单处理语句组成；程序从第一个语句开始，逐个语句按行顺序执行，直到 END 结束，没有分支和程序控制转移的情况出现。

与立即方式不同，程序方式需要：(a)输入、编辑程序。在编辑窗口中书写(键入)源程序(一行一个语句)，源程序输入、编辑修改过程只是将源程序装入内存储器送存，屏幕上显示程序，计算机没有执行程序。(b)运行程序。源程序输入后，在背景窗命令行下达系统命令 RUN 回车(或 F9)，开始运行程序，并在背景窗显示运行结果。运行结束命令行又出现 OK.—，等待以后的系统命令。

② 源程序设计及输入。用 True BASIC 语言求解任何问题的过程是：数据输入—数据处理—结果输出。这也是编写程序的逻辑思想。

在进入 True BASIC 后，按 F1 键光标进入编辑窗，并在第一个程序语句行行标(一个小方块■)后出现，等待输入 BASIC 语句。如果编辑窗和内存中有剩余的程序，首先应在命令行下达 NEW 命令，清除编辑窗口和内存储器中的剩余程序，再按 F1 键，进入编辑窗口。每键入一个字符，显示该字符，光标右移一列，回车换行键(ENTER)是约定的语句行结束符，每当输入一个语句，按回车换行键，编辑窗口出现下一个语句行行标，光标位于行标后等待新的一个语句输入。……直到全部源程序输入结束。

当源程序输入、修改后，需运行程序时，按 F2 键，光标进入背景窗的命令行，键入 RUN (或按 F9 功能键)命令，系统从第一行开始执行程序，在背景窗显示程序运行的结果。

③源程序编辑、修改

在源程序输入时，出现一些句法或程序逻辑错误是难免的，True BASIC 系统允许在整个编辑窗中编辑、修改源程序。一个源程序好似一篇文章，有字符、单词、行、块(或段落)、页(或窗)之分。修改可能出现在某页(或窗)，某块某行的某个单词，或某个字符，亦有可能是整行、整块的移动、拷贝插入或删除，用光标控制修改。True BASIC 语言具有强有力的全屏幕编辑功能，供源程序编辑、修改用。本节介绍单词、字符等小范围简单的编辑功能。

修改要使用光标控制键，首先将光标移到要修改的字符处，再作修改。

④ 语句行的删除、增加、分裂和合并

语句行删除：光标移到要删除行的行标上，按 DEL 键，删去当前语句行。

空语句行插入：光标移到要插入的语句行首部，按回车键(ENTER)，在当前程序行前插入一空行，光标在空行首部，等待输入语句。输入完毕，如果继续插入，按回车键，如不再插入，不按回车键，继续移动光标、修改，因为回车键总要产生一个新语句行。

语句行分裂：光标移到语句行需分裂处，按回车键，分裂成两个语句行。

语句行合并：欲将两个语句行合并时，将光标移至合并行的行标上，按退格键，该行连接到上个语句行的尾部，合并成一行。

长语句行输入和阅读：True BASIC 的语句行最多允许有 32767 个字符，远远超出屏幕 80 列的宽度。在输入超出 80 个字符时，无须采取任何措施，光标自动右移，左侧部分被卷出屏幕，直到输入完毕。在长行阅读时，可利用以上介绍的光标控制键，虽然屏幕上只显示 80 个字符，利用←/→键逐个字符向左/右推进，阅读，修改语句行的任何部分。利用 CTRL

+←/CTRL+→可迅速将光标移到语句行首部/尾部，只显示前/最后的 80 个字符。

表 10.1　光标移动控制键

键　符	光标控制作用
←/→	光标向左/右移动一个字符
TAB/SHIFT+TAB *	光标移到下/前一个字的开始字符
CTRL+←/CTRL+→	光标移到当前行的首部/尾部
↑/↓	光标上/下移一个语句行
PGUP/PGDN	光标移到当前窗的上/下一个窗(页)
HOME/END	光标移到源程序的首部/尾部
INS	字符插入/覆盖开关，缺省为插入状态

(1)表中除有 * 标注的键在标准打字机键盘上外，其它键均在计算器小键盘上，参看附录 1。

(2)INS 插入/覆盖开关，True BASIC 语言系统启动后，处于插入状态，光标为一，键入字符光标前插入。按一次 INS 开关，光标变为小方块■，改为字符覆盖状态，键入字符覆盖光标处字符。再按一次 INS 开关，恢复缺省的插入状态。

表 10.2　删除控制键

键　符	功　能
←退格键 *	删除光标前的一个字符
DEL	删除光标所在的字符
CTRL+HOME/CTRL+PGUP	删除单词的光标左/右侧部分
ESC/CTRL+END	删除语句行光标左/右侧部分
F7(RESTORE) * *	恢复当前修改部分

表中 * * 标注的键为功能键，其它同前。

(2)实验目的

① 掌握 True BASIC 语言简单程序编辑和运行。

② 掌握各种形式 PRINT 语句数据输出位置的控制方法。

③ 学习三种数据输入语句。

(3)实验范例

如鸡兔同笼等许多实际问题，实质是二元一次方程求解，可以归结成一个数学模型：

$$\begin{cases} AX+BY=E \\ CX+DY=F \end{cases}$$

利用项列式，直接可写出求解公式：

$$X=(ED-BF)/(AD-BC)$$

$$Y=(AF-CE)/(AD-BC)$$

例如鸡兔同笼，两者头数为 40，脚数为 100，求解鸡、兔各多少只。

(4)上机步骤

① 输入、编辑源程序

```
DATA 1,1,4,2,40,100
READ A,B,C,D,E,F
LET X=(E*D-B*F)/(A*D-B*C)
LET Y=(A*F-C*E)/(A*D-B*C)
```

```
PRINT  "Rabits=";X,"Cocks=";Y                END
```

② 运行程序

```
OK. RUN
Rabits=10        Cocks=30
OK.
```

③ 程序运行时出错处理。解释执行方式程序运行时检查语句句法和语义，一旦出现句法错误、变量类型不匹配，或无数据可读等错误时，自动终止程序执行，光标停留在出错的语句行，背景窗显示出错信息，等待修改。错误修改后，按 F2 键，如还有错误，光标自动移到第二个错误出现的语句行等待修改，直到全部错误修改完毕，再按 F2 键，光标停留在命令行时，再次运行程序。

(5)上机练习题

① 输入一个同学政治、英语、数学三门课考试成绩，编程，按下列要求输出该同学成绩报告单：

```
                              Statement of student
NAME:
POLITICS     ENGLISH     MATHMATICS     AVERAGE     TOTAL
```

②. 编程求解 $X^2+2X+1=0$ 的两个不等实根。

(6)思考题

①什么是程序？程序方式和立即方式有什么不同？

②归纳 PRINT 语句数据输出位置控制的各种方法。

③归纳三种数据输入语句的各自的优缺点。

10.2.3 判断选择结构

(1)预习知识

① 分支选择结构判断逻辑条件成立与否，控制程序选择执行相应的语句块。条件是一个比较关系式或由关系式和逻辑运算符构成的逻辑表达式，表达式只可能有真或假两种结果。

单行 IF/THEN 语句只能选择执行一个语句，且只能占用一个语句行，无需 END IF；判断选择结构允许选择执行语句块，此时必须有 END IF 结构框。

② 程序文件处理命令。存储在磁盘上的程序称程序文件。程序文件保存着 True BASIC 的源程序或编译程序，分别称源程序文件和编译文件。程序文件由 True BASIC 系统命令管理，允许装入内存并运行，或在编辑窗中显示、修改、再存储。编译文件不能显示、修改。有关程序文件更详细的叙述见 7.2 节。

③ 源程序打印输出和 DO 命令

当前内存中存储的文本打印前，可用 DO 命令预处理，用 LIST 命令打印或用 DO PAGE 命令分页打印。

DO 预处理命令有：

DO FORMAT 命令将当前源程序整理成 True BASIC 的标准格式，关键字大写、凹入结构等，以便打印或存储。

DO NUM 命令将源程序加上行号，如果当前程序已经有行号，则重新设置行号。当程序较短从 100 号开始，以 10 递增，如果大于 90 行，则从 1000 号开始。

表 10.3　程序文件处理命令

命令	命令功能说明
SAVE<文件引用名>	将当前在内存中的程序写到指定的磁盘上，建立由文件引用名命名的新文件
REPLACE	将当前在内存中的文本存盘替代装入文件
REPLACE<文件引用名>	将当前在内存中的文本存盘替代同名文件
OLD<文件引用名>	把当前盘上的指定文件装入内存
OLD<文件名.TXT>	把当前盘上扩展名为.TXT的指定文件装入内存
OLD<B:文件引用名>	把B盘上的指定文件装入内存。
COMPILE	编译内存中存储的源程序
FILES	显示当前盘上扩展名为.TRU的全部文件目录。
FILES *.*	显示当前盘上的全部文件目录。
FILES B:*.*	显示B盘上的全部文件目录。
UNSAVE<文件引用名>	从磁盘上删除文件引用名指定的文件。
NEW	清内存和编辑窗
RUN	运行内存中的程序
LIST	打印输出内存中的源程序或文本
LIST BLOCK	打印当前文本由块标设定块的部分内容
LIST n1-n2	打印当前文本n1-n2行的部分内容

DO RENUM，FIRST，STEP 命令源程序重新设置行号，FIRST 是指定开始行号，STEP 指定递增量。

DO UNNUM 命令删去源程序的行号，如果行号有一空格也被删去。

DO PAGE 命令当前文件分页打印输出，每页有页标，有标题，打印日期和时间。

(2)实验目的

① 掌握判断选择 IF/THEN 结构的句法和正确使用。

② 学会源程序文件建立，装入，简单修改，运行和再存储。

(3)实验实例

编程求解一元二次方程 $ax^2+bx+c=0$ 的两个根。

① 编辑源程序

```
INPUT PROMPT ″a,b,c=″;a,b,c
LET d=b*b-4*a*c
IF d=0 THEN
  PRINT ″x1,x2=″;-b/(2*a)
ELSE
  IF d>0 THEN
    PRINT ″x1=″;-b/(2*a)+sqr(d)/(2*a),
    PRINT ″x2=″;-b/(2*a)-sqr(d)/(2*a)
  ELSE
    PRINT ″x1=″;-b/(2*a);″+i″;
    PRINT SQR(-d)/(2*a),
    PRINT ″x2=″;-b/(2*a);″-i″;
    PRINT SQR(-d)/(2*a)
  END IF
END IF
END
```

② 运行程序

```
OK. RUN
a,b,c=1,2,1
```

```
x1,x2=-1                                x1=-2.5+i5.45436
OK.RUN                                  x2=-2.5-i5.45436
a,b,c=1,5,36                            OK.
```

③ 建立 Equal 源程序文件

```
OK. SAVE B:Equa
```

④ 清除内存

```
OK. NEW
```

⑤ 装入 EQUAL 源程序文件，再运行

```
OK. OLD B:Equal                         x1=-1+i 8.24621
OK.RUN                                  x2=-1-i 8.24621
a,b,c=1,2,69                            OK.
```

⑥ 替换存储

```
OK.REPLACE
```

⑦ 打印输出源程序

```
OK.LIST
OK.
```

(3)实验习题

① 改用 ELSE IF、SELECT CASE 结构改写实验范例。

② 一周岁以下婴儿身长可按下式测算

$$\begin{cases} L=L0+M*2.2 & M<=6 \\ L=L0+M*2.5 & 12>=M>6 \end{cases}$$

其中，L0 为出生时身长，M 为月龄。

③ 键入三个数 A，B，C 编程按大中小排列输出。

④ 编程计算

$$\begin{cases} Y=LOG(X)+SQR(X) & X>=0 \\ Y=ABS(X)*SGN(X) & -1<X<0 \\ Y=INT(X+.5) & X<=-1 \end{cases}$$

⑤ 编程实现润年推算。

(4)思考题

① 如何理解判断选择结构是单输入口，单出口的结构？

② SELECT CASE 结构是否完全可以替代 IF/THEN 结构，为什么？

③ 分析归纳各类判断选择结构。

④ 为什么要建立源程序文件？

10.2.4 **循环结构** 1

(1)预习知识

① 循环结构。循环是程序运行时控制转向前面语句执行，构成某个语句块反复执行的环路。True BASIC 有计数循环 FOR/NEXT、条件循环 DO/LOOP 两种结构：FOR/NEXT 由循环变量控制循环执行，DO/LOOP 由条件短语 WHILE、UNTIL 控制循环执行。无休止的循环称死循环，程序中不允许出现死循环，一旦出现死循环的错误，利用复合键 CTRL+BREAK 中止程序运行。程序设计中的穷举算法和迭代、递推算法实现工具是循环结构，循环充分发

挥了计算机低能、高速的特性。

② 文本编辑命令。文本编辑命令不仅可对当前源程序进行功能更强，更灵活的编辑工作，同样亦可以编辑正文文件。许多编辑命令系统已经定义在功能键上，如表 10.4 所示。详细功能结合命令介绍。

表 10.4　文本编辑命令

功能键	命　令	命令功能说明
F1		光标移入编辑窗口
F2		光标移入背景窗口
F3	FIND	只能在编辑窗用，寻找某个单词或数
F4	MARK	编辑文本作块标记
F5	COPY	编辑窗：在当前行后复制块标记部分的文本 背景窗：将当前行命令在命令行复制 命令行：在命令行复制最后输入的命令
F6	MOVE	将作了标记的文本，移到当前行之后
F7	RESTORE	最后的删除或插入修改无效
F9	RUN	运行当前程序
F10	HELP	获得系统帮助

(a)FIND 命令(F3)：在编辑窗中寻找当前文本某个单词或数。当光标在编辑窗中时，按 F3 键，此时在背景窗中显示 FIND 命令，键入要寻找的字/数，按回车键，系统从光标处开始寻找指定的单词或数字，并将光标停在第一个要找的单词/数字的首部。再按 F3 键，继续寻找下一个，……直到当前文本结束，再也没要找的单词/数字时，背景窗显示 NO FOUND，寻找结束，出现 OK. 提示的命令行。

FIND 命令不能直接在命令行键入，只能在编辑窗用 F3 键。

(b)LOCATE 命令：寻找、定位当前文本某个单词或数。在命令行键入：LOCATE 字/数时，系统将当前编辑文本中包含指定字/数的所有行，显示在背景窗口中。

(c)CHANGE 命令：寻找、修改当前文本某个单词或数。在命令行键入：CHANGE 旧字，新字时，系统将当前编辑文本中所有旧字/数字，全部用命令中指定新字/数字替代，背景窗显示修改的数量。

以上命令中的字/数是指整个单词/数值，并且大小写等同看待。如果指定的是单词/数字的一部分，则字/数应用双引号包括。此时，大小写字符有别，例如：

FINDTHE	寻找到当前编辑文本中的 THE 单词
CHANGE“10”，“20”	当前编辑文本中的 10 改成 20，101 改成 201
CHANGE10，20	当前编辑文本中的 10 改成 20，101 不变
CHANGE“THE”，“NEW”	当前编辑文本中的 THE 改成 NEW，THEN 改成 NEWN，OTHER 则改成 ONEWR

(d)TRY 命令：寻找、征询修改当前文本某个单词或数。TRY 命令和 CHANGE 命令非常相似，在命令行键入：TRY 旧字，新字后，系统在当前编辑文本中寻找指定的旧字，并用反相高亮度显示，在背景窗中显示 CHANGE？征询是否用新字替代，按回车（或键入"Y"）表示替代，按“N”是不替代。接着寻找、征询下一个指定的旧字，直到当前编辑文本结束，或键入“q”

作征询回答，结束 TRY 命令。

(e)EDIT 命令：用来指定或消去寻找搜索和修改命令的作用范围。

未加限制时，搜索和修改命令的作用范围是整个编辑文本。EDIT 命令指定块作为编辑限制范围，所谓块是由当前编辑文本(源程序或文本)的若干行组成。块可由 EDIT 语句指定，例如命令行键入：EDIT 行 1，行 2 将当前编辑文本从行 1 到行 2 的所有行指定成一块，此时编辑窗只显块的各行，系统的编辑作用亦限制在块的范围。对源程序文件，块可以是程序模块，此时 EDIT 命令可直接用子程序、子图、函数名限定编辑范围。如果命令行键入的 EDIT 命令未指定块，系统消去编辑限制范围，恢复整个文件的编辑作用。

(f)MARK 命令(F4)：MARK 命令给当前编辑文本作块标记，以便系统作块移动、删除、拷贝操作。当光标在编辑窗时，光标移至块的首行按 F4 键，该行行标为反相高亮度显示，再将光标移至块的末行按 F4 键，此时，从首行到末行全部行标为反相高亮度，块标记完毕。在背景窗的命令行，键入：MARK 行 1，行 2 命令，编辑窗从行 1 到行 2 所有行标作标记。同样，源程序模块可以用 MARK 命令后跟子程序，子图、函数名来标记块。在完成块编辑作后，块标记自动清除。要在未作块编辑清除块标记，只要将光标移到块中，按 F4 键就可。

(g)COPY 命令(F5)：COPY 是标记块拷贝命令。在编辑窗：将光标移到插入复制块的前一行，按 F5 键，则块被复制插入。在命令行，键入：COPY 命令是将标记块在当前编辑文本之后复制插入；键入：COPY 块 1 命令是将标记块在块 1 之后复制插入；键入：COPY 块 1，块 2 命令是将块 1 在块 2 后复制。

(h)MOVE 命令(F6)：标记块移动命令。在编辑窗：按 F6 将标记块移到当前行之后。在命令行，键入：MOVE 命令将标记块移到当前编辑文本之后；键入：MOVE 块 1 命令，将标记块移到块 1 之后；键入：MOVE 块 1、块 2 命令，将块 1 移到块 2 之后。

标记块的左右移动：按一次>键，整个标记块右移一列，按<键，标记块左移一列。

(i)TO 命令：光标在当前编辑文本行间移动命令。命令行键入：TO 块 1 命令，光标移到块的首行；命令行键入：TOn，对没有行号的编辑文本，光标移到第 n 行；对有行号的文本，光标移到行号 n 的行。可以用函数、子程序、子图名作为指定的块。如果 TO 后未指定块或 n，则光标移到当前编辑文本的第一行。

(j)KEEP 命令：除标记块或指定块外删除编辑文本所有其它行。命令行键入：KEEP 块 1 命令，除块 1 外，删除其它行，其中块 1 可以是函数、子程序、子图模块，或一个行数，或是一个行数的范围，如(100～210)。

该命令非常有用，在希望当前编辑文本中某块单独存贮时使用，但要注意，因为当前编辑文本中其它部分将均被删除。

(k)DELETE 命令：删除标记块或指定块。命令行键入：DELETE，删除标记块部分；键入：DELETE 块 1，删除块 1，其中块 1 可以是函数、子程序、子图模块、或一个行数，或是行数的一个范围。

(l)INCLUDE 命令：将指定的 ASCII 码磁盘文件插入当前编辑文本。命令行键入 INCLUDE 引用文件名，将指定的磁盘的文件插入当前行之后。所以在下达该命令前，在编辑窗将光标移到插入的前一行上。

(2)实验目的

①掌握循环 FOR/NEXT、DO/LOOP 结构的句法、正确使用实现迭代算法。

②学会文本块编辑、磁盘文件插入。

(3)实验范例

编程求解：
$$s=\sum_{i=0}^{n}2^i=2^1+2^2+\cdots+2^n(n=63)$$
$$\begin{cases}t=1, & s=0 \quad (i=0);\\ t=t*2, & s=s+t \quad (n<=i>0);\end{cases}$$

①编辑源程序

```
INPUT PROMPT ″n=″;n
LET s, t=1
FOR i=1 TO n
  LETt=2*t
  LETs=s+t
NEXT i
PRINT ″s=2^ 0+2^ 1+…+2^ ″;n;″=″;s
END
```

②运行程序

```
OK. RUN
n=63
s=2^ 0+2^ 1+…+2^ 63=1.84467e+19
OK.
```

(4)实验习题

①改用 DO/LOOP 结构改写实验范例。

②编程计算：M!，M=100。

③编程计算：$e^x=1+\frac{x}{1!}+\frac{x^2}{2!}+\frac{x^3}{3!}+\cdots+\frac{x^n}{n!}+\cdots$

要求用两种方法实现近似计算：(a)n=20；(b)公项<=10^6。

(5)思考题

①如何理解循环是单入口，单出口的结构？什么是循环结构的非正常出口？

②FOR/NEXT 结构是否完全可以替代 DO/LOOP 结构，为什么？

③归纳各类循环结构。

10.2.5 循环结构 2

(1)预习知识

①建立用户自己定义的语句关键字键

为方便源程序输入，避免在输入时出现语句关键字拼写错误，系统允许用户利用 KEY 命令，定义自己的语句关键字键。将一些常用的关键字用 ALT 键加字母双键定义。

KEY 命令：允许在键盘上重新定义一些特殊功能键(不可显示的字符键)的功能，但字母、数字、特殊符号等可显示的符号键不允许重新定义，往往是用复合键(双键)定义 True BASIC 的语句关键字和常用的系统命令。以下是将 ALT+I 复合键定义成判断选择结构 IF THEN/ELSE/END IF 的过程：

```
OK. KEY                                 命令行键入命令
KEY TO REDEFINE:ALT+I                   按 ALT+I 键作重新定义响应
DEFINE IT AS:IF THEN/ELSE/END IF        输入定义功能,CTRL+A 作为结束
OK.
```

例中/是回车键。定义后，在源程序编辑时，按 AIT+I 键，编辑窗出现三个分离行，分别是 IF THEN，ELSE 和 END IF。

系统允许把所有键的重新定义存储到文件中，使用：KEY TO<引用文件名>命令，也可以在需要时，将文件的键定义装入内存，使用：KEY FROM <引用文件名>命令。KEY 命令只能在命令行下达。

②程序运行结果打印机输出

系统将输入输出设备视为特殊的正文文件，当需要打印程序输出结果时，首先应用 OPEN 语句打开打印机，指定与打印机相关连的通道：

OPEN #<通道>:PRINTER

其中，<通道>为数值表达式，取值范围是 1～999。

打印机通道打开后，带通道的 PRINT 语句在打印机输出，

PRINT #<通道>:[<表达式>][{,/;[<表达式>]}]

程序运行结果在打印机输出，详细情况可参考 7.4 节。

(2)实验目的

①掌握循环结构嵌套和穷举算法。

②学会语句关键字键的重新定义，语句关键字键文件的建立，装入和使用。

③学会打印源程序的运行结果。

(3)实验实例

编程求 3～100 中的素数。采用双重循环结构嵌套：内循环采用穷举算法，判断数 i 是否为素数。i 逐个用 2～INT(SQR(i))的数除，如果能被其中的数整除，不是素数，否则为素数。外循环是 3～100 被判断的数。

①编辑源程序

```
PRINT "Display(0) or Print out (1-999)?"
INPUT ch
IF ch<>0 THEN OPEN #ch:PRINTER
FOR i=3 TO 100
    LET e=INT(SQR(i))
    LET m=2
    DO
      IF MOD(i,m)=0 THEN
        EXIT DO
      ELSE
        LET m=m+1
      END IF
    LOOP UNTIL m>e
    IF m>=e+1 THEN PRINT #ch:i;
NEXT i
PRINT #ch:
END
```

②运行程序，显示、打印源程序的运行结果

```
OK. RUN
Display(0) or Print out(1-999)? 0
3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97
OK. RUN
Display(0) or Printout(1-999)? 1
```

打印源程序的运行结果：

```
OK.
```

(4)实验习题

①全用 FOR/NEXT 或 DO/LOOP 结构改写双重循环结构嵌套，打印源程序、和程序的运行结果。

②编程输出以下图案：

(a).

```
    =
   ===
  =====
 =======
=========
 =======
  =====
   ===
    =
```

(b).

```
*          *          *          *
* *        * *        * *        * *
* * *      * * *      * * *      * * *
* * * *    * * * *    * * * *    * * * *
* * * * *  * * * * *  * * * * *  * * * * *
```

(5)思考题

①如何理解多重循环结构嵌套？

②如何理解穷举、迭代和递推算法？

③为什么系统的输入输出设备可视为特殊的正文文件？如何打印源程序？如何打印程序的运行结果？

10.2.6 数组

(1)预习知识

①文本模式。IBM－PC 系统具有八种不同特性的显示模式。对单色显示卡和显示器只能使用(“MONO”)单色文本模式；彩色图形接口卡(CGA)和彩色显示器可以使用“40”、“80”模式。“80”彩色文本模式一屏显示 25 行，每行 80 个彩色字符。在文本模式下字符是显示的基本单位，不能作图，用于书写行结构的源程序、数据和散文，或由程序 PRINT 语句输出数据和文本。文本模式前景可供选用的颜色见表 10.5。

表 10.5 文本模式前景可供选用小颜色

编号	色名	说明	编号	色名	说明
0	BLACK	黑	8	/	灰
1	BLUE	兰	9	/	亮兰
2	GREEN	录	10	/	亮录
3	CYAN	青兰	11	/	亮青兰
4	RED	红	12	/	亮红
5	MAGENTA	洋红	13	/	亮洋红
6	BROWN *	棕	14	YELLOW *	黄
7	WHITE	白	15	/	亮白

文本前景色设置语句格式为：

SET COLOR<字串表达式/数值表达式>

背景可供选用的颜色八种(0～7)，字符背景色设置语句格式：

SET BACK<字串表达式/数值表达式>

文本窗口背景色不允许设置。字符可以闪烁，闪烁设置语句格式：

SET COLOR<字串表达式>

其中，<字串表达式>——“字符前景色名/背景色名/闪烁”，例如，SET COLOR“RED/BLUE/BLINK”字符红色，背景兰色，闪烁；或将背景色设置为 8～15，可以使字符闪烁。

②全屏幕文本输出。在背景窗中显示程序运行结果简单、方便，但输出只能按行格式，字符没有彩色，不能满足如表格、菜单等格式化输出要求。“80”模式下，允许在全屏幕上按随意格式输出文本。执行以下语句系统均可置成“80”模式：

SET MODE″80″

```
CLEAR
SET MODE "TEXT"          * 2.03 版本
```

执行前面语句自动清屏。全屏幕光标设置语句为：

SETCURSOR＜行＞,＜列＞

其中,＜行＞,＜列＞为数值表达式,设置光标行、列值,PRINT 语句在光标位置输出文本。若未经模式设置,执行光标设置语句时,也自动清屏置成“80”模式。光标关闭/恢复语句为：

SETCURSOR"ON/OFF" 其中,"OFF"关闭光标,恢复光标。

(2)实验目的

①掌握数组的定义和使用。

②学会全屏幕彩色文本输出,显示程序运行结果。

(3)实验实例

随机产生 50 个 1～100 的数,全屏幕显示非排序的结果,采用泡泡排序,输出排序结果。

①编辑源程序

```
RANDOMIZE
DIM x(50)
FOR i=1 to 50
  LET x(i)=INT((RND * 100)+1)
NEXT i
CLEAR
SET COLOR 4
SET BACK 9
PRINT "unsort array x:"
MAT PRINT x
FOR s=LBOUND(x) to UBOUND(x)-1
    LET fl=0
    FOR i=LBOUND(x) to UBOUND(x)-s
        IF x(i)>x(i+1) THEN
           LET t=x(i)
           LET x(i)=x(i+1)
           LET x(i+1)=t
           LET fl=1
        END IF
    NEXT i
    IF fl=0 THEN EXIT FOR
NEXT s
SET BACK1
PRINT "sorted xrrxy x:"
MAT PRINT x
END
```

②运行程序,考察全屏幕格式化文本输出的程序运行结果。

(4)实验习题

①按降序排序改写程序,用全屏幕彩色文本方式显示程序运行结果。

②随机产生 50 个 10～100 的数,查找是否有数 56,查找成功,则显示“SUCCES”,否则显示“FAIL”。要求:用顺序查找;对分查找两种算法分别实现。

③编程输出 10 行的 PASCAL 三角形(见右图)。

```
                  1
               1     1
            1     2     1
         1     3     3     1
      1     4     6     4     1
   1     5    10    10     5     1
1     6    15    20    15     6     1
```

(5)思考题

①什么是数组？数组有什么作用？为什么说数组是结构类型数据？

②为什么要调整数组？如何调整数组？数组测试函数有什么作用？

10.2.7 函数

(1)预习知识

①程序中错误类型。程序中出现的错误有两类:运行态错误和逻辑错误。运行态出错包括:句法错误——违背 True BASIC 语法规定的错误,如非法变量名,语句关键字拼写错误,

丢失语句中必要分量，结构语句不匹配，数组、函数未经说明先使用等等；语义错误——句法符合规定，但违背了处理规则，如被零除溢出，负数求平方根、求对数，非方阵求逆，变量与数据类型不一致，文件类型不符，记录长度不匹配，访问方式不对等等。解释方式工作的系统，在程序运行时检查语法和语义错误，一旦发现错误，中止程序运行，指出错误语句行，提示出错信息。运行态出错可以在系统帮助下得到纠正。程序逻辑错误是指算法实现错误，甚至算法设计错误，使得程序运行得出非预期的结果。如循环控制变量或使用变量在循环中赋初值，循环终止条件是某个实数等式，致使出现死循环或结果错误；又如丢失输出语句，致使无结果输出；逻辑颠倒、混乱，运行结果出错等等。True BASIC 系统对逻辑错误无能为力，只能通过程序调试发现、排除。

②程序调试。程序调试是项仔细的工作和反复的过程，没有一定的规章可循，原则上是尽可能利用系统提供的软件工具，通过实践积累经验。

一般采用分段跟踪调试程序。在调试前最好通读程序，了解程序要解决什么问题，如果程序包括函数、子程序或子图（总称过程），应该了解什么地方调用过程，过程参数是什么等信息以便跟踪时参考。在程序中有意识地插入 PRINT 语句可以找出这类信息，输出过程或循环开始、结束的特殊变量，指示程序执行流程和重要变量的状态，配合人工逐条执行程序语句，对小型程序往往行之有效的方法。

(a)断点设置。True BASIC 源程序无论是强迫中断、出错中止或是运行终止，程序中变量维持当前状态。True BASIC 提供三种中断程序运行方法。

强迫中断：程序运行时按 CTRL+BREAK 复合键。由于程序中断位置不可控制，往往用于运行时紧急处理，如强迫中断死循环，中止无用输出等等。

STOP 语句：有意识地在程序中插入 STOP 语句分段，由于不能继续运行，并要改动程序仍不方便。

BREAK、CONTINUE 命令：不必改动程序，又能有意识设置断点，配合语句命令可有效地监视变量状态，分段调试程序。

BREAK 为断点设置或去消命令。用 BREAK 命令设置断点时，首先将光标移到编辑窗设为断点的行标上，用 F4 键作块标记，然后光标移入背景窗命令行，键入 BREAK 命令，此时编辑窗标记行出现一个小三角形，表示该行设置成断点；命令行：BREAK 块，表示在指定块的第一行插入中断点；重复设置一次可消去设置的断点。系统允许设置多个断点，程序运行到断点暂停。

CONTINUE 为被 BREAK 命令中断执行的程序继续运行。该命令不能继续运行终止执行的程序；出错中止，或修改了错误的程序也不能用它继续运行。

BREAK，CONTINUE 配合实现程序分段调试。

(b)自动跟踪 TRACE 调试工具。2.03 版本提供 TRACE 调试软件，它由 DO 命令调用，格式为：

```
DO  TRACE,<方式>(<变量表>)
```

其中可指定的方式是“STEP”单步、“SLOW”慢速、“FAST”快速和“BREAK”断点四种中的一种，<变量表>指定要跟踪的变量，多个变量用逗号分隔。

执行该命令时，自动清屏并置成全屏幕文本方式，调试的源程序在屏幕左侧显示，右侧分成两个窗口，上部窗口显示跟踪变量的当前状态，下部窗口显示程序的当前输出；按指定

的方式执行程序，并用高辉度显示当前执行的语句，指示程序执行流程。注意被跟踪的变量不得超过 8 个。程序跟踪执行过程中，允许用单键改变跟踪方式："S"改为慢速方式；"F"快速方式；空格键为单步方式；"Q"停止跟踪。

在复杂程序联调时，过程的参数和变量一样被跟踪，但必须指明参数所属的过程，即用：过程名:参数

如果已经确信过程是可靠的程序模块，联调时可以不对过程跟踪，命令格式：

DO TRACE，<方式>(<变量表>)|(<过程>)

其中<过程>指定不被跟踪的函数、子程序或子图。

(2)实验目的

①掌握函数的自定义方法和正确调用。

②进一步提高编程和调试程序的能力。

(3)实验实例

随机产生长度为 10 的大写英文字母串 a$，键入待查字母 b$ 及起始查找位置 n，利用外部自定义函数 POS 输出查找结果。

函数 POS(a$，b$，n)：在 a$ 字符串的第 n 个字符开始查找子串 b$，返回子串第一次出现的位置值；若 n<=0，则取 n=1；若 b$ 长度大于 a$ 长度，或 n 大于 a$ 长度，返回 0。

```
DECLARE DEF pos
RANDOMIZE
FOR i=1 TO 10
  LET a$ =a$ & CHR$(INT(RND * 25)+65)
NEXT i
PRINT a$
DO
  INPUT b$,n
  PRINT pos(a$,b$,n)
LOOP UNTIL b$ ="0"
END
DEF pos(a$,b$,n)
  IF n<=0 THEN LET n=1 ELSE
    LET n=int(n)
  LET l1=LEN(a$)
  LET l2=LEN(b$)
  IF l2>l1 OR n+l2-1>l1 THEN
    LET pos=0
  ELSE
    FOR i=n TO l1-l2+1
      IF a$[i:i+l2-1]=b$ THEN
        LET pos=i
        EXIT FOR
      END IF
    NEXT i
    IF i>l1-l2+1 THEN LET pos=0
  END IF
END DEF
```

(4)实验习题

①解决实例中的几个问题：通过分段跟踪调试，排除错误；改写程序忽视 b$ 的大小写；输出所有子串出现位置值；将 POS 函数改为内部自定义函数。

②编写一个程序，输入两个直角边长，返回直角三角形斜边长度，直到用户不想算为止。要求用以下两种自定函数返回斜边值：(a)内部函数：利用全程变量传递直角边；(b)外部函数：利用参数传递直角边。

(5)思考题

①什么是内部函数，外部函数？各有什么特点？

②函数根据什么分类？

10.2.8 子程序

(1)实验目的

①掌握子程序编写方法，带参子程序设计及调用方法。

②学习递归算法编程。

(2)实验习题

①编写阶乘子程序，利用子程序调用计算：

$$C = \frac{m!}{n!(m-n)!} \quad (m > n, \text{且 } m, n \text{ 为正整数})$$

②编写一个程序，输入两个直角边长，返回直角三角形斜边长度，直到用户不想算为止。要求用以下两种子程序计算斜边值：(a)内部子程序，利用全程变量传递直角边和斜边值；(b)外部子程序，利用参数传递直角边和斜边值。

(3)思考题

①归纳总结函数与子程序的不同？何谓值参数、变量参数？

②子程序参数传递与函数有何不同？

③递归算法的条件是什么？

10.2.9 文件

(1)实验目的

①掌握正文文件的添加、读出操作。

②掌握记录文件建立和随机访问操作。

(2)实验习题

建立一个单位职工情况的数据文件(包括编号、姓名、年龄、职称、工资、文化程度等)，编写简单管理程序。

①采用正文文件，要求有下列模块：建立职工情况正文文件，人机对话方式输入 5～10 个职工情况，职工添加，按姓名查询，显示输出。

②采用记录文件(单字段或多字段)，要求下列模块：人机对话方式建立记录文件，记录添加，记录修改，记录按姓名查询，显示输出。

(3)思考题

①正文文件、记录文件各有什么特点？适用什么场合？

②为什么记录文件具有随机访问特性？

③如何用字节文件构成相应的数据结构？

10.2.10 图形

(1)实验目的

①掌握作图语句，学习在窗口中输出图形、文本的方法。

②掌握多窗口作图技术。

(2)实验习题

编程将屏幕划分为左右各半的两个窗口，左测窗口作图，右侧窗口显示菜单，分别建立正弦、余弦、李沙育作图模块，由右窗选择控制左窗口作出相应图形。

(3)思考题

①什么是图形显示方式，什么是文本显示方式？

②子图几何变换有几种？如何作动画？

10.2.11 True BASIC 与原 BASIC 语言的区别

(1)True BASIC 和原 BASIC 语言的区别

①True BASIC 语言无需行号，并且一行只能有一个语句，语句长度最长可达 32767 个字符。不允许在一行中有冒号分隔多个语句。简单语句占一行，但控制结构可以占多行。

True BASIC 的一行由语句和回车换行符两部分组成，回车换行符标志语句结束，在源程序输入时是必不少，虽然回车换行符在屏幕上并不可见。语句是程序行的核心，每个语句是 True BASIC 能识别的一条指令。简单语句以语句关键字开始，后跟语句体。关键字往往是命令计算机执行不同操作的动词或介词，语句体是操作对象。关键字和语句体等语句分量间必须有空格分隔，否则会引起句法出错。

②True BASIC 中赋值语句的关键字 LET 不能省略；程序终止的 END 语句不可缺少，一个程序单位只有最后的程序行是一个 END 语句，标志程序单位结束。

③注释语句的关键字 REM，可以用！替代，但程序行的注释只能以！开始的字符串。

④True BASIC 的数值无整型、单精度和双精度之分；字符串最长可达 32767 个字符；字符串的连接操作符是 &，不能用＋号。

⑤True BASIC 中无 GOTO 语句的结构化程序设计语言，程序控制结构的非正常出口使用各种 EXIT 语句。

其它版本的 BASIC 程序可以在 True BASIC 下运行，具体方法：

①其它版本的 BASIC 程序必须以 ASCII 码存储；

②在 True BASIC 下(如果使用汉字，设置成高分卡模式)调入，运行程序；

③在 True BASIC 下存贮程序，自动改成 True BASIC 程序。

(2)True BASIC 语言系统辅助命令和命令使用

辅助命令：辅助命令均在命令行输入。

ByE 命令：退出 True BASIC 系统，返回 DOS 状态。使用该命令前应检查当前文件是否已经存储，否则将会丢失。

HELP 命令：获得 True BASIC 系统命令的解释，HELP TOPICS 命令可以看到所有能解释的系统命令。HELP 命令也可按 F10 键获得，而 F10 键更灵活，因它可以在任何时刻使用，甚至程序执行期间，响应输入过程也允许使用。

True BASIC 语言系统命令的使用：所有 True BASIC 系统命令都可以用前三个或更多字母的缩写形式表示。如 REPLACE 可以写成 REP 或 REPL，但是不能在超过三个字母后的字母出错，例如 REPLACE 写成 REPQ 为非法命令。

允许在一个命令行下达多个系统命令，命令间用分号"；"分隔，称单批命令。TrueBASIC 系统自左而右逐个执行，执行完时再出现 OK. 提示，如果其中某个命令有错，则抛弃剩余的命令。单批命令只能是系统命令，语句命令只能单个执行。

附录 1　ASCII 码符号集

十进制	名称	十六进制	十进制	名称	十六进制	十进制	名称	名字	十六进制
000	nul	00	043	+	2B	086	V		56
001	soh	01	044	,	2C	087	W		57
002	stx	02	045	—	2D	088	X		58
003	etx	03	046	.	2E	089	Y		59
004	eot	04	047	/	2F	090	Z		5A
005	enq	05	048	0	30	091	[		5B
006	ack	06	049	1	31	092	\		5C
007	bel	07	050	2	32	093	]		5D
008	bs	08	051	3	33	094	^		5E
009	ht	09	052	4	34	095	—	UND^{+}	5F
010	lf	0A	053	5	35	096	`	GRA	60
011	vt	0B	054	6	36	097	a	LCA	61
012	ff	0C	055	7	37	098	b	LCB	62
013	cr	0D	056	8	38	099	c	LCC	63
014	so	0E	057	9	39	100	d	LCD	64
015	si	0F	058	:	3A	101	e	LCE	65
016	dle	10	059	;	3B	102	f	LCF	66
017	dc1	11	060	<	3C	103	g	LCG	67
018	dc2	12	061	=	3D	104	h	LCH	68
019	dc3	13	062	>	3E	105	i	LCI	69
020	dc4	14	063	?	3F	106	j	LCJ	6A
021	nak	15	064	@	40	107	k	LCK	6B
022	syn	16	065	A	41	108	l	LCL	6C
023	etb	17	066	B	42	109	m	LCM	6D
024	can	18	067	C	43	110	n	LCN	6E
025	em	19	068	D	44	111	o	LCO	6F
026	sub	1A	069	E	45	112	p	LCP	70
027	esc	1B	070	F	46	113	q	LCQ	71
028	fs	1C	071	G	47	114	r	LCR	72
029	gs	1D	072	H	48	115	s	LCS	73
030	rs	1E	073	I	49	116	t	LCT	74
031	us	1F	074	J	4A	117	u	LCU	75
032	sp	20	075	K	4B	118	v	LCV	76
033	!	21	076	L	4C	119	w	LCW	77
034	"	22	077	M	4D	120	x	LCX	78
035	#	23	078	N	4E	121	y	LCY	79
036	$	24	079	O	4F	122	z	LCZ	7A
037	%	25	080	P	50	123	{	LBR	7B
038	&	26	081	Q	51	124	|	VLN	7C
039	'	27	082	R	52	125	}	RBR	7D
040	(	28	083	S	53	126	~	TIL	7E
041	)	29	084	T	54	127	del		7F
042	*	2A	085	U	55				

注：$^{+}$当用到这个符号以下的各名字的名称，必须引用大写字母。

附录2　出错信息代码及解释

一般情况下 True BASIC 处理的错误信息可分为下述三类：(1)命令错。(2)编译态出错。(3)运行态出错。

当给 True BASIC 的命令不正确时，就会发生命令错。当 True BASIC 编译源程序时，所出现的错误称为编译态错。当程序开始运行时，产生的错误称为运行态错。在此附录中，对于各种出错信息给出有关解释(有的出错没有标出出错号)。

(1) Argument types con't match　当带参调用子程序时，若在该子程序未定义之前，或者是使用不同于原定的参数来调用该子程序时，会出现此错，即：调用的参数个数不对，或是类型不一致，此时应核对一下子程序中参数变量的定义及调用时所使用的相应值。

(2) Bad FIND item ; try using quotes　包含逗号或双引号的符号串必须用双引号括住(此规则在 INPUT 语句和 DATA 语句中是一样的)。

(3) Badly formed USING string (8201)　在 USING $ 函数或 PRINT USING 语句中使用的格式串有错，至少有一项不符合 True BASIC 规则。

(4) Badly formed input line (3105)　在回答 INPUT 语句（用于读磁盘或读键盘)时，格式不正确，最常见的错误是丢失了双引号。

(5) Can't continue　通常可以用 CONTINUE 命令使暂停的程序继续运行。但是，True BASIC 不能继续运行该程序的可能的原因是：根本未执行过该程序，或再次运行之前修改了此程序，或当程序是由于出错而停止运行的；或在使用处 CONTINUE 命令之前用过 DO 命令。

(6) Can't copy region itself　True BASIC 的编辑程序不允许复制自身的程序行。例如，把某个程序再复制到那个子程序的本身之中。如果非要这样做，应先把子程序复制到别处。改名后，再复制到原来的子程序中来，然后再做其它整理。

(7) Can't do graphics on this computer (11000)　计算机不能画图，是由于它没有图形卡，或者没有使用相应的 SET WINDOW 、PLOT、DRAW、BOX、FLOOD、GET POINT 或 GET MOUSE 语句。

(8) Can't edit compiled program　当程序被编译以后就不能再被修改了。一旦使用过 COMPILE 命令之后，就只能使用 RUN 、SAVE 或 REPLACE 命令来处理这一程序了。如果还想改动，应事先留有副本，来编辑以前的副本。

(9) Can't erase file not opend as OUTIN (7301)　如果文件没有用 ACCESS OUTIN 打开，是不能使用 ERASE 语句的。

(10) Can't find library name　在使用 LIBRARY 语句时给出的文件名在当前目录中不存在。应检查确认记录此文件的磁盘是否插入计算机。

(11) Can't help with that , try HELP TOPICS　当求助时，或用 HELP 命令或用 F10 键。True BASIC 如不能对你所提的问题给予帮助。首先应保证拼写正确(不能缩写)，其次，要求 True BASIC 盘在机器中\TBHELP 子目录中有相应文件。键入 HELP TOPICS 将会列出 True BASIC 可以给出解答所有项目的列表。

(12) Can't input from OUTPUT file (7303)　用 ACCESS OUTPUT 打开文件是不能用来读取数据的。如果你非要读，则应选择项改成 ACCESS OUTIN。

(13) Can't invert singular matrix(3009)　当使用矩阵函数 INV 时，若打算对奇异阵施行求逆是不行的。奇异阵在 True BASIC 中规定不得求逆。

(14) Can't output to INPUT file (7302)　ACCESS INPUT 打开的文件，不能写入数据，如果非要写，应把选择项改成 ACCESS OUTIN 。

(15) Can't PRINT to middle of text file (7350)　正文文件中不得重写数据，要写时要先用 RESET 语句把文件指针移到文件的末尾再写，或者删除文件内容后才能重写。

(16) Can't SET RECSIZE on non empty record file (7250)　当某一记录文件被建立后，它就具有固定的记录长度。在改写或重读此记录文件时不能改变文件的记录长度。

(17) Can't SET WINDOW in picture(11004)　子图是不能重新设置窗口或屏幕坐标的，亦不能使用 OPEN SCREEN 或 SET WINDOW 语句。

(18) Can't use ANGLE(0,0)　(3008) 不能定义 ANGLE(0,0)，其中的参数至少要有一个是非零的。

(19) Can't use this statement here　在使用 True BASIC 语句结构时，不能写错位置。例如 SELECT CASE 语句中的 CASE 写错了位置或丢失了，IF—THEN 语句中用错了 ELSE IF，SELECT CASE 与第一个 CASE 之间多加了语句，等等。

(20) Can't use #0 here(7002)　在 OPEN 或 CLOSE 语句中不能使用 #0 通道，因为 #0 通道总是打开的。

(21) Channel is already open (7003)　试图打开的通道已经打开了。在程序中应检查在别处是否使用这一通道，所以要记住：不使用的通道应及时关闭。

(22) Channel isn't a window (11005)　当使用 WINDOW 语句打开或关闭某个窗口时，所给通道已与一个文件相连（不是某个窗口），再核对一下 WINDOW 语句中所给的通道号是否正确。

(23) Channel isn't open (7004)　当用某个通道来访问文件或窗口时，相应的通道没有打开。在使用通道时，必须先用 OPEN 语句打开它。核对一下所给定的通道号是否正确，以及它是否真的打开了，是否用 CLOSE 语句顺便关上了此通道。

(24) Channel number must be 1 to 1000 (7001)　通道的数值必须选取在 1 到 1000 范围内。修改你的程序，以使通道号不超出此范围。

(25) Compiled program　这是一个提示显示，它出现在给出 COMPILE 命令之后，或调出一个编译过的程序时(在编辑窗口上显示)。由于编译程序把源程序代码整理成内部形式，故不能再看到编译前的程序正文，也别打算再修改，否则就应先留有副本，再编译或根本别编译。

(26) Canstant too large：constant　对计算机来说，被显示的常数值太大而无法处理，可用 PRINT MAXNUM 看一下计算机所能表示的最大数值。此时应考虑修改有关常数。

(27) Cursor set out of bounds (8601)　SET CURSOR 语句给出的行和列的位置，超出当前窗口范围。应记住，每个窗口有它自已的光标坐标。每一窗口左上角位置总是行为 1，列为 1。可用 ASK MAX CURSOR 来找出最大的行、数值。

(28) Data item isn't a number (8101)　企图从记录文件中读取一个数，但文件的当前记录项却是一个字串，可能定位有问题。

(29) Data item isn't a string(8014)　试图从记录文件中读取一个字符串，但文件当前记录却是一个数。

(30) DET needs a square matrix (6002)　Det 函数仅用于方阵，这是受数学定义所限。

(31) Disk full (9006)　当把文件逐个向磁盘写入时，所占的磁盘空间就越来越大，直至不能再装，就是满了。这种错一般由操作系统直接给出。

(32) Diskette remove or wrong diskette。(9005)　已经打开了某个文件，True BASIC 已在使用它。可是你却把磁盘取出换上了其它盘，所以出错。正在使用的磁盘不要从驱动器中取出。

(33) Division by zero (3001)　表达式中用零来除某一数值。

(34) Doesn't belong here　在程序中，光标所处的位置有错误。检查程序中使用的语句并改正之。

(35) Endig doesn't match beginning　使用的结构语句，例如 FOR—NEXT 或 IF—THEN—ELSE 必须开头与结束相呼应。通常是忘掉了结束语句，或者是在循环语句中头、尾的下标变量不相同。应仔细地阅读这些程序行，找出不匹配的地方。

(36) Error in PLAY string (4501)　在 PLAY 语句中所给定的字符串不符合 True BASIC 文法。

(37) Expected "thing"　当前光标所指向的位置上不是 True BASIC 允许的字符或标点符号，而是一些其它的东西，这一信息对于重新修补某语句是有益的。

(38) Expected relational operator　当前光标所指的位置上必须有某个关系运算符，例如"＝"或"＜"等等，程序于此必须写完某种比较运算（注意，True BASIC 不允许用"IF A THEN…" 形式的语句而应改为" IF A＜＞0 THEN…"格式）。

(39) File already exists (9004)　当建立某一新文件时，实际上它已存在。应该对一下有关的文件名，如果打算重写某个已存在的文件，那么在 OPEN 语句中把 CREATE NEW 应改写成 CREATE NEWOLD 或者要用 REPLACE 命令，代替 SAVE 命令。

(40) File is read or write protected (9001)　当试图读、写某文件时，操作系统对此文件防读或防写。True BASIC 是不会处理防读和防写的，所以应退出 True BASIC 解决操作系统对此文件所做的防读和防写动作。

(41) File pointer of bounds (7252)　当打算用 RESET 或 SET POINTER 或 SET RECORD 语句来改变文件的指针，但是现在所给定的文件指针不是比 1 小，就是超出了文件结束符，可用 ASK FILESIZE 查出文件有多长。

(42) File's record size doesn't match open recsize (7103)　记录文件都有它的 RECSIZE 建立相应的记录长度。当被打开的文件的 RECSIZE 与在 OPEN 语句中给出的 RECSIZE 不一致时出现问题。如何才能正确地打开此文件呢？简单的方法是删除 OPEN 语句中的 RECSIZE 部分而仅使用记录长度。若还想在 OPEN 语句中使用 RECSIZE ，那么可先执行一小段检测程序，查出文件的 RECSIZE，方法是不用 RECSIZE 打开文件，用 ASK RECSIZE 语句确定它。

(43) IND must make a squaqe matrix (6004)　矩阵必须是方阵。另外当使用 IND(X,Y) 函数时，必须 X＝Y。

(44) ILLegal array bounds for name in routine　在数组定义中，上界小于下界是非法的。核对相应 DIM 语句所设置的数组（实际上，True BASIC 允许下界比上界大 1 ，用于定义一个不含元素的数组，但是比 1 大是不行的）。

(45) Illegal channel　在使用 SET 或 ASK 语句时，在应给出所需的通道号，而语句中没有给出或在 SET 或 ASK 语句中不允许有通道号时却加入了通道号。

(46) Illegal data　这是 DATA 语句书写不当，逗号只能放在两个数据项之间，而不能放在某个数据清单的末尾。所有的双引号必须成对书写。

(47) Illegal exit　EXIT DO 语句写在 DO 循环之外，EXIT FOR 语句写在 FOR 循环之外，或写入一个未知类型循环的退出语句，如 EXIT FED，确保 EXIT 语句在相应结构中且后附字正确，主要是 EXIT 语句的后缀。

(48) Illegal expression　光标所指的表达式中有不符合 True BASIC 规则的地方，应检查一下是否用了双操作符（"1＋＋2"），或在一个数字中加入了逗号（例如，"1,1000"），另外所使用的变量名是否合法，等等。

(49) Illegal keyword　光标指出的关键字在此处用法不正确。例如，在 BOX 语句中是否丢失了 LINES、AREA、ELLIPSE 或 CLEAR 关键字。检查相应语句，并改正它。

(50) Illegal number　光标所指位置需要一个数字，但没有给出，还可能是所书写的数字不符合 True BASIC 文法。DIM 和 OPTION BASE 语句中只能用整数常数。

(51) Illegal option　True BASIC 只提供了两种选择语句为 OPTION ANGLE 和 OPTION BASE。可能是 ANGLE 、BASE 、DEGREES 及 RADIANS 拼写有错。

(52) Illegal parameter　当使用 SUB DEF 或 PICTURE 定义一个子程序时，在参数表中的某一参数可能有错，如使用了同名参数，或使用了与简单变量名同名的参数。

(53) Illegal statment　True BASIC 的每一种语句都必须在开始带有关键字，象 LET SELECT 等等。核对一下所拼写的关键字是否正确。有些 BASIC 版本在书写赋值语句时可以省掉 LET 关键字，但 True BASIC

是不允许省掉的。

(54) Improper ORD string(4003) ORD 函数所需要的是一单字符串，就是由 ASCII 码组成的合法字符串。

(55) Improper NUM string (4020) 给出的 NUM 函数串不能表示为 IEEE8 八字节的浮点数。检查一下建立时是否正确。

(56) Input item bigger than RECSIZE (8302) 此条信息仅说明损坏的记录文件中某些记录要比文件定义的记录长度要大。如果磁盘上包含有且用到这种带变异性的文件此信息就会出现，或者没有使用 True BASIC 文件，而是干预了其它不明了的字节文件时，也有可能发生。

(57) INV needs a square matrik (6003) 矩阵求逆只限于方阵。当试图用 INV 函数转换某个非方阵时，是不允许的。应检查一下矩阵是否两维，且每维是否都有相同的长度。

(58) LBOUND out of range (4008) 当用 LBOUND(A,3) 调用时，数组 A 不是三维以上的。检查一下相应的维数使用情况。

(59) LOG of number <= 0 (3004) 算术对数自变量不能取负值。

(60) MARGIN less than zonewidth(4006) SET MARGIN 语句中所设置的 margin 比 zonewidth 小是不允许的，应检查一下相应的变量值。

(61) Mismatehed array sizes (6001) 在使用 MAT 语句是需要相同大小的数组，但使用的数组大小却不同，例如矩阵加法就需要两个数组有相同的大小，矩阵乘法就要有更复杂的规则。

(62) Missing end statment 整个 True BASIC 程序必须包含有 END 语句，但当前丢掉了，在合适的位置上加上它再试一下。

(63) MOD and REMAINDER can't have 0 as 2nd argument (3006) MOD 和 REMAINDER 不允许零作为它们的第二个参数，因为这是等于用零来除某数。应仔细检查一下所用的参数。

(64) Must be a function name 当书写 DEF 或 FUNCTION 语句时，没有在相应关键字后面跟有适当的函数名。

(65) Must be a number True BASIC 几乎在任何可用简单数值的地方允许使用数字表达式，但有些地方例外，例如，CASE 就不能用表达式，只能用数字值常数。如果必须用表达式，那就应重写这个 SELECT CASE 结构，而改成 IF—THEN—ELSE 结构。

(66) Must be a picture name DRAW 语句中用的不是一个子图名字。应改成一个子图名再试。

(67) Must be a string constant True BASIC 在可以使用字符串常数处用字串表达式取代，但有少数例外，如 CASE 测试就不能用串表达式，如果非要用，只能把 SELECT CASE 结构改成 IF—THEN—ELSE 结构。

(68) Must be a subroutine name CALL 语句仅能用于调用子程序，所给的串必须是子程序名，改写语句使用正确的子程序名。

(69) Must be a variable 在应当使用变量的地方，使用了表达式或子程序名，例如 LET 、INPUT 和 GET 语句就一定要用变量。检查一下这些语句使用的是否正确，是否与子程序、函数、子图重名。

(70) Must be a array 在 True BASIC 的程序中，很多地方都必须使用数组变量，而不是某一简单变量，如 MAT 语句仅对数组有效。许多函数如 LBOUND 和 SIZE 也只能对数组有效。因此应核对一下数组的名字是否拼写正确，是否与 DIM 语句中定义一致。

(71) Must be byte file or empty for SET RECSIZE (7251) SET RECSIZE 仅能用于字节文件，或空白的记录文件。正文文件不能有记录长度，非空的记录文件有它自已的记录长度。检查一下所使用的通道是否正确。

(72) Must be record or byte file for SET RECORD (7202) SET RECORD 不能用于正文文件。可以重新设置一正文文件的指针指向文件的最开始或文件的末尾。

(73) Must be record or byte file (8502) 当使用 READ 或 WRITE 语句是不能用于正文文件或一个窗

口，它们只能工作在记录和字节文件，核对一下所使用的通道是否有上述情况。对于正文文件及窗口只能使用 INPUT 和 PRINT 语句。

(74) Must be text file (8501)　当使用 INPUT 或 PRINT 语句是不能用于记录文件或字节文件，它们只能工作在正文文件中。记录文件或字节文件应使用 READ 或 WRITE 语句。可用 ASK ORGANIZATION 询问一下文件的类型，再检查是否使用正确。

(75) Must SET RECSIZE before WRITE (8304)　在往一个空记录文件上写入时，没有定义文件的记录长度。在第一条 WRITE 语句之前必须执行 SET RECSIZE 语句，或用 OPEN 语句来指定文件的记录长度。

(76) Name can't be redefined　对任何两个不同内容的事物均不可使用同一个名字，比如，如果变量名定义为 X ，那么子程序名就不能是 X，对于重名 True BASIC 总是判出错。

(77) Negative number to non—integral power (3002)　当计算 N 的 X 次幂时，N 为负数且 X 为非整数，其结果在数学上无意义。

(78) No CASE selected, but no CASE ELSE (10004)　在执行一 SELECT CASE 语句时，CASE 测试不成功，因为没有 CASE ELSE 语句块来获取这一问题。True BASIC 就显示上述信息。检查一下选择表达式是否使用得完整，若加上 CASE ELSE ，在测试时可处理各种情况的问题。可试一下。

(79) No GET MOUSE on this computer (8700)　企图用 GET MOUSE 语句，但在计算机上并没有加上一鼠标定位装置。

(80) No saved copy—still want to compile ?　当给过 COMPILE 命令之后，True BASIC 将要抛弃当前程序内容，而以不可见的编译代码记盘。为了提醒你注意先做好必要的复制工作，特显示此信息。"N" 为退出此命令；"Y"为真正开始编译。故在回答时，应小心地检查一下是否已存盘，或不打算留下副本。

(81) No such color(11008)　SET COLOR 语句所设置的颜色名，True BASIC 不认识。色彩名可用大写，也可用小写，但不能在名字中使用多余的符号。再用 True BASIC 能确认的颜色名完整清单，检查一次程序。

(82) No such file (9003)　要打开的文件实际上不存在，从很多命令中也可能得到这个信息（例如 OLD）。首先应检查一下名字是否用对了，再看有关的磁盘是否插入到驱动器里了。可用 FILES 命令配合检查一下当前的磁盘内容。

(True BASIC 也象一些操作系统一样使用了扩展名，例如，所有的 True BASIC 程序都自动地加上了" TRU" 的扩展名，这一点也应引起注意。)

(83) No such function or subroutine　在某命令中给出了某个函数、子程序或子图的名字，但这些子程序却不存在。例如，可键入 LIST MYFUNC ，而名字也确实拼写正确了，但 MYFUNC 不存在。

(84) No such line numbers　命令中所给出的行号与实际程序中的行号范围不符。

(85) No USING item for output (8202)　在使用 USING $ 函数或 PRINT USING 语句时，应有格式描述串。但是这里没有，而仅有的是有关的正文。

(86) Not found　可以使用 FIND 键来查找某些字或短语，但并没有找到。记住，True BASIC 匹配整字。如果不打算找一整字，象"print"，而是要找"i"，那么必须用双引号括住所要查找的有关内容，如下：

Find:"i"

这样，True BASIC 将找出 "PRINT" 中的 "i"。还要注意，True BASIC 总是从当前光标位置开始一直找到文件末尾。如果要找光标前的内容，移动光标。

(87) ON index out of range , no ELSE given (1000)　当用了 "ON GOTO" 或 "ON GOSUB" 语句时，所指定的分支号出界了，例如，若 X 比 1 小或比 3 大。而用 "ON X GOSUB 100,120,130" 时，就会出现这样的信息，应核对一下相应下标变量的值。

(88) Out of memory (5000)　这时，问题出在要求提供的存储器比当前机器上所支持的还要多。由于 True BASIC 将使用计算机所提供的全部存储器，这只有通过购买存储器来解决问题了。另外也应做一些必要的在小存储器环境下的实验，建议如下：①采用小数组。多维数的数组可简化成一维数组来分块处理。②

编译源程序。因为 True BASIC 的编译代码要远比源程序代码短小得多。③检查一下所有的子程序，能采用递归调用的尽量写成完美而合法的递归子程序，这是常用且有效的方法。④如果是一位高级程序员，可以使用一下 Packb 和 Unpackb 函数作为最后的手段，它可以压缩数据而少占内存。

(89) Output item bigger than RECSIZE (8301)　当往一记录文件中去写某数据项时，由于文件建立时设置了固定的记录长度，而不能写入超长的数据。可用 ASK RECSIZE 来找出文件的记录长度。记住，任何数据项均不能超过记录长度的指定值，每个数字只能是 8 字节长。

(90) Overflow (1000)　当要计算的数值超出计算机所能处理的范围时就产生溢出。PRINT MAXNUM 可见到计算机所能使用的最大数。如果打算通过溢出来获得尽可能大的数值，而又不出错，可写成：

```
WHEN ERROR IN
     LET X=TAN(PI/2)
USE
     LET X= MAXNUM
END WHEN
```

(91) Please say what lines you mean　假如键入了 MOVE 命令，但又没有指出所要移向的行位置，可用 MARK 键(F4) 来标记行，或者重新键入带上移向行位置的 MOVE 命令。

(92) Please try "CHANGE old ,new"　当修改一短语时，必须同时给出原来的短语和要改的短语。各短语中若包含有逗号或双引号时，都必用双引号括住其整个内容。

(93) Please try "DO filename"　当用 DO 命令时，必须给出文件名，重新键入带文件名的 DO 命令！

(94) Please try INCLUDE filename　当用 INCLUDE 命令时，必须给文件名，重新键入带有文件名的 INCLUDE 命令！

(95) Please try "KEY" or "KEY FROM file" or "KEY to file"　KEY 命令有三种形式。"KEY"本身是恢复原来键的含义，"KEY TO"和"KEY FROM" 让你更改键的含义，可以由一个文件保存键定义或恢复键定义。

(96) Please try "OLD filename"　当使用 OLD 命令时，必须给出所要装入的文件名，以便 True BASIC 从磁盘中调出此文件。

(97) Please try "SAVE filename" or "REPLACE filename"　当你将当前文件存盘时，必须给出文件名。

(98) Please type line numbers as 100 or 100—150　假如所给出的命令是 DELETE，应该带有的参数是一个行号或由行号表示的行模块。不带后面这些说明项 True BASIC 就不懂了。DELETE 100 是删除第 100 行。DELETE 100—150 是删除 100 到 150 行的程序块。

(99) Program stopped　按 CTRL+BREAK 键，程序就被中止运行。程序执行中，不按此键无法停止运行。

(100) REPEAT $ count<0 (4010)　在使用 REPEAT $ (A $,N)函数时，N 不能小于零的数，检查一下此处所使用变量的名是否正确。

(101) Reading past end of data(8001)　在执行 READ 语句时，所读的 DATA 项比给出的 DATA 项要多，造成读出界。解决的办法有两种：其一，是使读的项与定义的数据项在个数上应取一致；其二，读之前可用 MORE DATA 测试一下，再决定是否继续读。

(102) Reading past end of file　当试图从文件中读取多于文件记录内容时出此错。适合于各种文件。防止的方法有两。其一，明了文件的大小及记录格式；其二，可用 MORE # N 和 END #N 预先测定相应文件是否结束了。

(103) RETURN without GOSUB (10002)　当执行 RETURN 语句时，可能没有对应的 GOSUB 语句事先执行过，所以出此错。诊断查找此错有时是很困难的。建议使用子程序名字调用方法，这样的结构清晰，而且不容易发生类似的错误。

(104) Screen bounds must be 0 to 1 (11003)　以 OPEN SCREEN 语句设置的屏幕边界必须在 0 到 1 的范围之内。不管屏幕有多大，其左下角定义为 0；右上角定义为 1。

(105) Screen minimun＞＝maximum (11002) OPEN SCREEN 语句用四个数定义新窗口。顺序为：左、右、上和下。右边界必须大于左边界；上边界必须大于下边界。如果打算获得相反的窗口，就不能使用此语句，而应使用 SET WINDOW 语句。

(106) SIZE index out of range (4004) 假如，使用的是函数 Size(A,3)，当数组 A 的维数小于 3 就会出此错。应检查 DIM 语句定义的数组是否正确。

(107) SQR of nagative number (3005) 不能对负数求平方根。

(108) Statement outside of program 光标当前所指的程序行已超出主程序，又不属于任一外部子程序、函数和子图。主程序一定要以 END 语句结尾。比如有一程序行在 END 语句之后，且又没有说明它是子程序或外部函数，则出错。

(109) String given instead of number (8103) 在执行 INPUT 语句输入某个数值时，没输入数值而输入了字符串，属类型错。如果是从文件读数据，应检查一下文件内数据类型与此语句中变量类型是否一致。如果是从键盘上读数据，且读的是个字符串而不是欲读的数值，可用 VAL 函数将它转换成数值。

(110) String too long (1051) 所建立的串长于 32000 个字节。

(111) Subscript out bounds (2001) 向数组中赋值时，超出了数组边界。可用 LBOUND 和 UBOUND 查出数组相应边界，且在屏幕上显示出来。

(112) TAB column not between 1 and margin (4005) 在 PRINT 语句中使用 TAB 函数时，所给定的参数不是 1 至当前允许范围之内，出此错。可用 ASK MARGIN 语句找出当前区界。

(113) The BYE command is just "bye" 当退出 True BASIC 时，只能键入"BYE"，而不能带有任何其它东西。

(114) The COMPILE command is just "COMPILE" 使用 COMPILE 命令时，也不能附带任何其它内容，只需打入"COMPILE"。

(115) The CONTINUE command is just "CONTINUE" 使用 CONTINUE 命令时，不能在此命令上附带任何其它内容。

(116) The RUN commandis just "RUN" 运行当前的程序时，只能键入"RUN"，不能附加任何其它内容。

(117) This must first appear in a DIM or DECLARE DEF 对于任何的数组和外部函数来说，必须先定义后使用。

(118) Too few input items (8002) 在执行 INPUT 语句时，输入的项数比在 INPUT 语句中要求的项数少。如果输入来自键盘，将要求继续输入。如果来自文件，就应核对一下文件中的项数与语句中的项数不一致的原因，然后做相应改正。

(119) Too many channels open (7102) 如果想打开十个以上的通道是不行的，True BASIC 在同一时刻只能处理十个通道。你应适当地关闭一些暂时不用的通道（某些操作系统仅可以同时打开小于十个的通道，可查一下用户指南看看计算机能打开多少）。

(120) Too many input items (8003) 在执行 INPUT 语句时，输入项数比在 INPUT 语句中要求的项数多。如果输入来自键盘，则忽略所有多余项。如果来自文件，就应核对一下文件中的项数与语句中的项数不一致的原因，然后做必要的改动。

(121) Trouble using disk or print (9002) True BASIC 在使用某个磁盘或打印机时发现问题。不同的计算机出错的原因也不尽相同。首先应检查一下电源是否接通，另外打印机的纸是否装好，它是否处于暂停状态，驱动器中是否没装磁盘，打印机电缆是否接好等等。如果始终找不到错，应在计算机上运行诊断故障程序来解决。

(122) Type is wrong for name in routine 当调用某个子程序时，用的是另一个程序的名字，这时将会出现相应参数不匹的类型错。否则就是把子程序中定义的参数，在调用时把数量与类型搞错了。数字参数变量不能赋给字符串，反之亦然。子程序不能作为函数使用，反之亦然。

(123) UBOUND out of range (4009)　假如使用了 UBOUND(A,3)，且 A 是一个小于三维的数组就出错了。检查一下 DIM 语句定义，再核对一下所使用的维数。

(124) Undefined routine name in routine　在某个子程序中，调用名代表的子程序、函数或子图，没有定义过。检查调用程序名是否正确或是否在 LIBRARY 语句中含有此程序。

(125) Unknown OPEN option (7101)　在 OPEN 语句后面的选择部分 ACCESS、CREATE 或 ORGANIZATION 没有给出。这种错误的出现有二：其一，是根本没有写上；其二，是可能拼写错或用缩写词不对。

(126) VAL string isn't a proper number (4001)　所用的 VAL 函数中，给定的串必须表示一适当的数。

(127) What ? (Please type HELP)　所键入的命令 True BASIC 不懂。如果想求助 True BASIC 来解释有关的使用，应键入 HELP，然后回车，计算机会告诉你简要的说明。此时，True BASIC 盘应插在磁盘驱动器。

(128) Window minimun=maximun(11001)　当使用某个 SET WINDOW 语句时，所设置的垂直、水平数值，最大值等于最小值。True BASIC 不允许这样，因为在这样的窗口里无法显示任何东西。可能问题出在把设置边界的顺序搞错了，应该是左、右、下、上。

(129) Wrong number of arguments　在调用某个子程序、函数或子图时，所给出的调用参数值有错。检查一下子程序相应的定义及调用的方法。

(130) Wrong number of dimensions　在使用数组时，维数给错了。核对一下 DIM 语句定义及使用的位置上的维数是否一致。如果是把一个数组传给某个子程序，那就要检查一下子程序的参数。记住二维数组在子程序参数表中必须表示成 A(,)；三维应为 A(, ,)等等。

(131) Wrong type　所定义的任何量必须与使用的任何量在类型上保持一致。如，串变量不能赋数值；反之亦然。记住，字符串进行连接时应书写"&"而不能使用"+"。

(132) Wrong type of file (7104)　True BASIC 所建立的各种文件均有它自已的类型标志。以前记录在磁盘上的文件应与用 OPEN 语句要打开的文件描述在类型上保持一致。虽然 OPEN 语句使用合法，但由于与打开的文件类型不匹配也是枉然。

(133) You have two routines called name in routine　子程序的重名定义为非法。对任何不同的子程序必须给于不同的名字，否则 True BASIC 不知道调用哪一个。这种人为的二义性容易避免，核对一下子程序名。

(134) Zero to negative power(3003)　当求 0 的 N 次幂时，N 不能小于零。这在数学上称为无意义。True BASIC 也会给出出错信息。

(135) ZONEWIDTH out of range (4007)　当执行 SET ZONEWIDTH 语句时，所设置的 zonewidth 比 1 小或比 margin 大。可用 ASK MARGIN 语句来找出当前的 margin。

Document generated by Anna’s Archive around 2023-2024 as part of the DuXiu collection (https://annas-blog.org/duxiu-exclusive.html).

Images have been losslessly embedded. Information about the original file can be found in PDF attachments. Some stats (more in the PDF attachments):

```
{
  "filename": "MTA5NDM1MjQuemlw",
  "filename_decoded": "10943524.zip",
  "filesize": 24062345,
  "md5": "6cf64ab355d6197b8a5878bac10f7b35",
  "header_md5": "93383b4f9afdc7f562ab23cbac763229",
  "sha1": "56c34d0c8fd65a234ea75020438f1a38ce4fb3b6",
  "sha256": "09854939671ad4526f308d8e0e2a9671966486d4b36bc3c8fd88d97f47a90704",
  "crc32": 1534733102,
  "zip_password": "",
  "uncompressed_size": 23946040,
  "pdg_dir_name": "\u255c\u00df\u2563\u2563\u2557\u00bb\u2502\u2560\u2568\u2265\u2554\u03a6\u255d\u255e\u255c\u2560\u2502\u2560\u00ed\u00ac\u00ed\u00acTrue BASIC_10943524",
  "pdg_main_pages_found": 242,
  "pdg_main_pages_max": 242,
  "total_pages": 249,
  "total_pixels": 1747669248,
  "pdf_generation_missing_pages": false
}
```